Live Memory Forensics of Mobile Phones

By

Vrizlynn Thing, Kian-Yong Ng and Ee-Chien Chang

From the proceedings of

The Digital Forensic Research Conference

DFRWS 2010 USA

Portland, OR (Aug 2nd - 4th)

available at www.sciencedirect.com

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

Live memory forensics of mobile phones

Vrizlynn L. L. Thing[a,*], Kian-Yong Ng[b], Ee-Chien Chang[b]

[a] Cryptography & Security Department, Institute for Infocomm Research, 1 Fusionopolis Way, #21-01, Connexis (South Tower), Singapore 138632, Singapore

[b] School of Computing, National University of Singapore, Singapore

Keywords:
Live forensics
Volatile memory
Mobile phones
Android

ABSTRACT

In this paper, we proposed an automated system to perform a live memory forensic analysis for mobile phones. We investigated the dynamic behavior of the mobile phone's volatile memory, and the analysis is useful in real-time evidence acquisition analysis of communication based applications. Different communication scenarios with varying parameters were investigated. Our experimental results showed that outgoing messages (from the phone) have a higher persistency than the incoming messages. In our experiments, we consistently achieved a 100% evidence acquisition rate with the outgoing messages. For the incoming messages, the acquisition rates ranged from 75.6% to 100%, considering a wide range of varying parameters in different scenarios. Hence, in a more realistic scenario where the parties may occasionally take turns to send messages and consecutively send a few messages, our acquisition can capture most of the data to facilitate further detailed forensic investigation.

1. Introduction

The standard digital forensic procedure typically involves steps such as "pulling the plug", acquiring data from the static media, analysing and correlating the data to retrieve the relevant evidence in a forensically sound manner (Ashcroft et al., 2004). This forensic investigation procedure prevents further interference on potential evidence, is well-documented and has proven to be reliable and hence, acceptable by the law enforcement agencies during crime investigation involving computer systems. However, measures to prevent interference of potential evidence may instead result in the undesirable loss of important evidence.

When the evidence is stored (or being transferred) off-site or a communication session discussing criminal activities is on-going, the "pull the plug" approach may not be an appropriate action. In addition, as the static storage media increases in size, so does the amount of acquired data and potential evidence that requires processing. A live analysis of the current (though dynamic and volatile) state of the system and its applications, is therefore necessary, so as to allow a more efficient forensic investigation process. In addition, techniques to protect the privacy of users and confidentiality of user data, such as encryption and password protection, have also indirectly provided counter forensics means to technologically aware criminals. Therefore, conventional forensic methods are no longer adequate and more research efforts have been placed in live memory forensic analysis of computer systems (Carrier and Grand, 2004; Adelstein, 2006; Petroni et al., 2006; Schatz, 2007; Kiley et al., 2008; Simon and Slay, 2009) in recent years to complement or enhance the former methods.

In mobile phone forensics, live memory forensics has an even more important role to play. As mobile phones are

* Corresponding author.
E-mail addresses: vriz@i2r.a-star.edu.sg (V.L.L. Thing), ngky@comp.nus.edu.sg (K.-Y. Ng), changec@comp.nus.edu.sg (E.-C. Chang).

1742-2876/$ – see front matter © 2010 Digital Forensic Research Workshop. Published by Elsevier Ltd. All rights reserved.
doi:10.1016/j.diin.2010.05.010

becoming increasingly prevalent and are constantly evolving into "smarter" devices (i.e. smartphones with higher processing power and enhanced features), capabilities to perform in-depth forensics on these devices also become essential. However, current mobile phone forensics are still restricted to the research and analysis of static data on the Subscriber Identity Module (SIM), memory cards and the internal flash memory (Willassen, 2003; Forensic analysis, 2006; Jansen and Ayers, 2006; Williamson et al., 2006; Casadei et al., 2006; Ayers et al., 2007; Kim et al., 2007; Al-Zarouni, 2007; Mokhonoana and Olivier, 2007; Jansen et al., 2008; Bhadsavle and Wang, 2008; Distefano and Me, 2008; Ahmed and Dharaskar, 2008; Berte et al., 2009; Hoog, 2009; Hoog and Gaffaney, 2009). Although the constraint on the storage capacity implies that they do not face the problem of exceedingly large amount of potential evidence to be analysed, it introduces another problem. Due to this limited storage, volatile information such as the application data, Internet browsing data, and instant messaging conversation histories are often not stored in the non-volatile storage media. This is unlike computer systems which allows the caching and backup of a large amount of data (e.g. MSN chat history). In this case, the limitations of static forensic analysis on mobile phones become even more evident. Without the means to perform live memory forensics on mobile phones, potentially incriminating evidence may be lost forever.

As a mobile phone's main functionality is to support communications, the capability to perform forensic analysis on its interactive based applications is very important. In this paper, we propose an automated system to support the mobile phone's live memory dynamic properties analysis on interactive based applications. We implemented the system components and performed an investigation on the persistency of the mobile phone's volatile data and real-time evidence acquisition analysis. The mobile phone used in our investigation was an Android mobile phone, the Google development set. The choice of the Android platform was due to it being the latest released mobile platform and its fast rising popularity among users and the mobile phone manufacturers (Kumparak, 2010). As our future work, we will be investigating on applying the methodology and porting the system to other mobile phone platforms.

The rest of the paper is organised as follow. In Section 2, we present an overview of research conducted on mobile phone forensics. We describe our live memory forensic analysis system in Section 3. The experiments and results are presented and discussed in Section 4. Future work is described in Section 5. Conclusions follow in Section 6.

2. An overview of mobile phone forensics research

In an early work (Willassen, 2003), Willassen researched on the forensic investigation of GSM phones. The author presented the types of data of forensic relevance, which can exist on the phones, the SIM and the core network, and emphasized the need for more sound mobile forensic procedures and tools. In (Forensic analysis, 2006), Willassen proposed extracting the physical image of the mobile phone's internal flash memory by desoldering the memory chip and reading it from a device programmer. Another proposed method was to read the memory through the boundary-scan (JTAG) test pins. The extracted memory was examined to detect the presence of deleted file contents.

In Casadei et al. (2006), the authors presented their SIMbrush tool developed for both the Linux and Windows platforms. The tool relied on the PCSC library and supported the acquisition of the entire file system, including the non standard files, on the SIM. However, files with restricted read access conditions could not be extracted.

In Kim et al. (2007), the authors presented a tool to acquire the data from a Korea CDMA mobile phone's internal flash memory. The tool communicated with the phone through the RS-232C serial interface and was able to acquire the existing files on the phone using the underlying Qualcomm Mobile Station Modem diagnostic mode protocol.

In Al-Zarouni (2007), the author studied the mobile phone flasher devices and considered their applicability in mobile phone forensics. As these devices offered access to the phone's flash memory and did not require installation on the phone, they were deemed to be forensically sound. However, their operations were not well-documented. Since they were designed to write to the memory, the effect of evidence altering while performing a read was unknown. Their reading capability and memory access range also varied for phones of different brands and models.

In Mokhonoana and Olivier (2007), the authors proposed an on-phone forensic tool to acquire the active files from a Symbian OS v7 phone and store it on the removable media. Instead of interfacing with the PC connectivity services, the tool interacted with the operating system to perform a logical copy of the files. The tested phone was Sony Ericcson P800. One main limitation of the tool was that those files in use could not be copied (e.g. call logs, contacts).

In Distefano and Me (2008), the authors proposed the mobile phone internal acquisition technique on the Symbian OS v8 phones. The mobile phone data was acquired using a tool residing on the removable media, instead of the PC/mobile phone USB connection based approach. The tool utilised the Symbian S60 File Server API in the read-only mode. The authors carried out experiments comparing the tool with Paraben Device Seizure (USB connection to phone) and P3nfs (Remote access through Bluetooth). The tool took a longer time to perform the acquisition. It was able to acquire more data compared to the P3nfs but lesser data compared to the Paraben Device Seizure. However, the authors observed that the larger data size from Paraben was due to the additional information from its acquired data management.

In Jansen et al. (2008), the authors proposed a phone manager protocol filtering technique by intercepting the data between the phone and the phone manager. The objective was to address the latency in the coverage of newly available phone models by existing forensic tools. The authors also proposed an identity module programming technique, to populate the phone's SIM with reference test data, so as to provide a baseline for the validation of SIM forensic tools.

2.1. *Surveys on existing tools*

In Jansen and Ayers (2006), the authors evaluated the state-of-the-art SIM forensic tools to understand the capabilities and limitations in their data acquisition, examination and reporting functions. The tools surveyed included Cell Seizure, GSM.XRY, MOBILedit! Forensic, TULP 2G, Forensic Card Reader, ForensicSIM, SIMCon and SIMIS. It was observed that most information such as the IMSI and SMS/EMS could be found by the tools.

In Bhadsavle and Wang (2008), the authors evaluated the effectiveness of the Paraben Device Seizure on a test data populated T-Mobile locked SIM. They determined that 100% of the test data were retrieved.

In Williamson et al. (2006), the authors studied the performance of different mobile phone forensic tools (i.e. TULP 2G, MOBILedit! Forensic, Cell Seizure and Oxygen Phone Manager) on the Nokia phones. The authors concluded that some tools failed to deliver some promised features (e.g. MD5 hash was not found in MOBILedit!, SHA1 hash was not found in Cell Seizure).

In Ayers et al. (2007), the authors conducted a comprehensive study on the current mobile phone forensic tools and presented their findings in the NIST report. The evaluated tools included the Paraben Device Seizure, Pilot-Link, GSM. XRY, Oxygen Phone Manager, MOBILedit!, BitPIM, TULP 2G, SecureView, PhoneBase2, CellDEK, SIMIS2, ForensicSIM, Forensic Card Reader, SIMCon and USIMdetective. The authors presented each tool's capabilities and limitations on a range of mobile phones, covering different operating systems, processor types, and hardware components. Some examples of the tested phones included Samsung SGH-i300, Motorola MPX220, Sony Ericsson P910a, Nokia 7610 and BlackBerry 7780.

In Hoog (2009), the author presented the existing forensic evidence acquisition tools for the Android phone. The Android Debug Bridge (ADB) enabled interaction with the phone over the USB connection. Therefore, active files on the phone can be retrieved through the "adb pull" command. Other tools such as the Nandroid backup and Paraben Device Seizure also supported the extraction of files residing on the phone.

In Hoog and Gaffaney (2009), the authors analysed the effectiveness of existing forensic tools' capabilities in acquiring information (e.g. call logs, SMS, contacts, Emails, image files) from a 3G iPhone. The phone was running firmware 2.2 and was not jailbroken. The tools analysed included WOLF, Cellebrite UFED, Paraben Device Seizure, MacLockPick, MDBackupExtract and Physical DD. Of all the tools tested, the DD approach was the only one allowing a bit-by-bit copy of the phone's storage and was ranked the highest (i.e. its results indicated that the acquisition of the test data sets either met or exceeded the expected results).

2.2. *Related work*

More closely related to our work, were the recent forensic investigations on the evidence data from the instant messaging (IM) services (Kiley et al., 2008; Husain and Sridhar, 2010). IM services can be accessed through applications installed on the PC (i.e. client-based) or web-based using a web browser.

In Kiley et al. (2008), the authors examined the artifacts that can be recovered from 4 web-based IM services (i.e. AIM Express, Google Talk (GTalk), Meebo and E-buddy) on a Windows XP system. The AccessData Forensic Toolkit was used to acquire a bit-stream image of the system hard drive and to perform an indexing of the image file. A keyword search on the distinct phrases used as the test data was carried out. The Runtime DiskExplorer was also used to examine the image file through a sector-by-sector keyword search. The objective was to find any residual data residing in the unallocated hard disk space. The authors found that while evidence of forensic value (such as screen name and estimated time of conversation) could be retrieved, very limited chat logs were recoverable. In addition, the authors only used a short list of unique phrases (e.g. "bannnnanas", "spaces spled wrong") as keywords in the experiments. Parameters such as the message length and human response time during chatting, which could affect the experimental results were not defined. Evidence acquisition without the knowledge of the messages contents was also not discussed.

In Husain and Sridhar (2010), the authors performed a forensic analysis of the AIM, Yahoo! Messenger and GTalk on the Apple iPhone, to investigate the possibility of IM related evidence recovery. The experiments were conducted on the client-based versions of AIM and Yahoo! Messenger, and the web-based version of GTalk. For the study, the test data was created by sending two consecutive messages for each IM. The data on the phone was acquired logically through the iTunes Backup. Forensically relevant evidence such as the screen names, timestamps and unique phrases (i.e. the chat messages) was found in the retrieved files for the AIM and Yahoo! Messenger. However, no trace of evidence was found for GTalk. The temporary Internet files and caches of the Safari browser (on which GTalk was accessed) did not contain much information other than the fact that GTalk was accessed at a particular time.

From the prior art, we observed that only the evidence residing in the phone's non-volatile storage was retrievable. The challenge arises when the application data presents itself in the volatile data only (e.g. in web-based applications) and no trace of evidence could be found in the non-volatile storage. We address this issue on mobile phone forensics in this paper.

3. Live memory forensics of mobile phones

In this section, we propose an automated system to investigate the dynamic properties of the mobile phone's volatile memory, and to perform a real-time evidence acquisition analysis. We used the Android phone as the test set. A few system components were developed specifically for the Android platform. An overview of our automated system is shown in Fig. 1.

Our system consists of the following components.

- Message Script Generator (MSG)
- UI/Application Exerciser Monkey
- Chat Bot
- Memory Acquisition Tool (*memgrab*)
- Memory Dump Analyser (MDA)

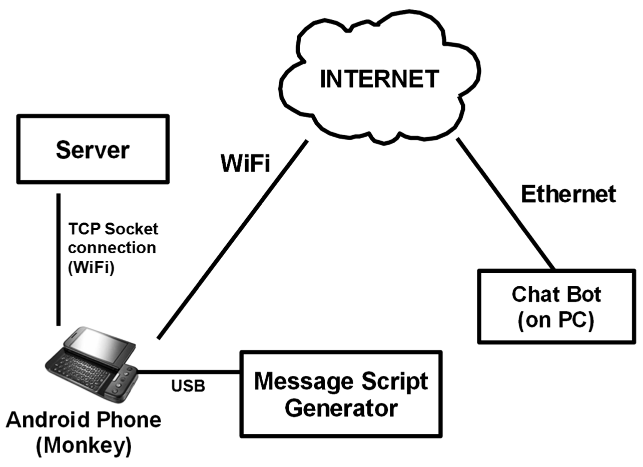

Fig. 1 – System overview.

3.1. *Message script generator (MSG) and UI/application exerciser monkey*

The Monkey is part of the Android software stack and is generally used to perform pseudo-random actions to the programs on the phone. Instead, we control the Monkey to instruct it to perform deterministic actions with our MSG, through the Android Debug Bridge (ADB). The MSG randomly generates a test message and a Monkey script during each experiment run, to be executed by the Monkey.

3.2. *Chat Bot*

To automate the chatting process with the phone, we developed a Chat Bot emulating a user at the PC. The Chat Bot is a Java console-based application that uses the Smack API. The Smack API is a Java library for IM clients to communicate using the Extensible Messaging and Presence Protocol (XMPP), which is also used by the GTalk service.

Configurable parameters on the MSG and the Chat Bot include the message length, the character set to support random message generation, and the interval between messages (to emulate typing and reading response time). The contents of each message are composed by randomly selecting characters from the specified character set. The MSG and Monkey also support configurable interval between keypresses.

3.3. *Memory acquisition tool (*memgrab*)*

Firstly, we look at how the process memory is represented in Android, to perform the process memory acquisition.

The process memory management in Android is handled by the Linux 2.6 kernel. However, Android uses the anonymous shared memory subsystem (ashmem) driver for handling the shared memory among processes, instead of the standard Linux kernel's IPC shared memory module (Chen, 2008). The former allows the shared memory blocks to be reclaimed by the kernel, whereas the latter forbids it. Android also replaces the Out-of-Memory (OOM) Killer module with its own Low Memory Killer driver to sacrifice the running processes on the phone when the memory is low.

In Android, each running process' memory information can be obtained from the procfs virtual file system provided by the kernel. When an Android application is started, its directory is created under/proc. These directories are named by their process IDs (/proc/pid). The process memory can be accessed from the/proc/pid/mem file, while the entries in the/proc/pid/maps file provide the addresses of the process memory regions. The memory regions compose of the stack, heap and shared memory and also contain mappings to the system libraries, Dalvik executables and device drivers.

We developed our memory acquisition tool, *memgrab*, which performs a dump (or "grabbing") of a process' memory. The tool locates and acquires the process memory by relying on the/proc/pid/maps and/proc/pid/mem files in the procfs. The idea is similar to the pcat tool, which is part of The Coroner's Toolkit (TCT) (Farmer and Venema, 1999). However, our *memgrab* tool is customised for the Android platform, and supports specific memory regions extraction and specification of the acquisition interval.

The *memgrab* tool performs the Process Trace (ptrace) system call, which allows the tracing of a process by controlling its execution, as well as gaining access to its address space. To commence tracing of the target process, PTRACE_ATTACH is specified in the system call. The target process is then suspended while *memgrab* acquires a snapshot of the process memory. PTRACE_DETACH is then issued to resume the target process' execution.

3.4. *Memory dump analyser (MDA)*

In the memory dump, the outgoing message appears in two forms (i.e. before and after processing to be sent as a network packet to the chat server).

```
[["m","userone@gmail.com/Smack
-> F0FBEF9E","MESSAGE","MSG_ID","c"]]
```

and

```
[[1011,["c",['4890438E28A0973D',["m","
-> userone@gmail.com/SmackF0FBEF9E"
->,"MESSAGE","","s",["MESSAGE"]
```

while the incoming message appears in one form:

```
[[1013,["c",['4890438E28A0973D',["m","
-> usertwo@gmail.com/SmackEC940444"
->,"MESSAGE","","r",["MESSAGE"]
```

We identify the common structure in the messages as:

```
["m","USER_ID@gmail.com/XMPP_RESOURCE",
-> "MESSAGE","
```

and use the following regular expression to capture the message contents.

```
["m","[.a-zA-Z\d]+@gmail.com
-> [/a-zA-Z0-9]*",",(.*);
```

Based on a list of the generated messages, the MDA consists of Perl scripts to perform searches on each dump. The results of the findings of whole or partial messages are then compiled into a report.

Other than the Chat Bot which is specific to message exchange applications using XMPP, the other components are generic and can be applied to any communication based applications running on the phone. In the next section, we focus on the investigation of the persistency of the relevant evidence pertaining to a web-based Google Talk (GTalk) IM. We also perform an investigation on the cached data to examine the presence of evidence relating to the chat application.

4. Experiments and results

In this section, we describe our experiments and present our results on identifying the memory region of a process where the message exchange can be observed, and investigating the cached data and the volatile evidence data persistency.

4.1. *Process memory region investigation*

To prevent dumping the irrelevant memory regions, we conducted an experiment to identify the region where the messages reside. In each experiment, we started a chat session between the phone and the PC. The phone sent 15 outgoing messages and received 15 incoming messages from the PC in return.

For each entry in the /proc/pid/maps, we performed a memory dump and searched for the occurrence of the messages in the dumps. Our experiments revealed that the messages were consistently found in the shared memory region.

We observed that the heap and stack memory contained information such as the database initializations and chat session credentials. However, such information has a higher persistence and can also be found in the cached data (Section 4.2). Next, we analyse the cache data and subsequently, focus on the analysis of the volatile evidence in the form of chat messages exchange.

4.2. *Cached data examination*

Before we proceed with the volatile evidence investigation, we first examine the browser cached data to find out what information we can retrieve. We conducted an experiment to exchange 15 messages between the two parties (i.e. Chat Bot and Monkey) before obtaining the cached data from the databases stored in the /data/data/com.android.browser/databases/ directory. There were 3 SQLite databases in the directory, namely browser.db, webviewCache.db and webview.db.

The browser database consisted of the bookmarks and searches tables. The bookmarks table stored the URLs that have been accessed in the past, while the searches table contained all the past keywords that have been entered to the search box.

The webviewCache database consisted of the cache table. The cache table held the cached images (e.g. gif, jpeg, ico, png), javascripts (js) and cascading style sheets (css).

The webview database consisted of the formdata, httpauth, cookies, formurl and password tables. The formdata table stored the saved values keyed into a form. The httpauth table held the past HTTP authentication information and was empty in our case. The cookies table stored the saved cookies. The formurl table stored the url of the past form the user has accessed. The password table stored the saved passwords. In our case, it contained only a single entry from accessing the WIFI router.

We concluded from this experiment that the exchanged chat messages were not retrievable from the cached data.

4.3. *Volatile evidence data persistency investigation*

Before proceeding with the experiments, we need to determine a realistic set of parameters such as the message length and the message sending interval. After which, we perform the investigations with different memory dump intervals and analyse the implications of the different parameters.

1) *Interval between Keypresses*: The faster a user types and sends more messages, the sooner the memory will be overwritten by new messages. For typing on the phone, we define it as the process of pressing the keys on the phone's QWERTY keyboard. The keyboard measures 6.8 cm from the Q to P keys. In (Sears et al., 1998), it was determined that a novice user can type on average, 9.9 ($\sigma = 2.4$) English words per minute (wpm), while an experienced user can type 21.1 ($\sigma = 1.5$) English wpm, on a 6.8 cm QWERTY keyboard (assuming 5 characters per word). Taking the worst case parameter for our experiments (and assuming a normal distribution), we assume that a user can type as fast as most experienced users (at 24.1 wpm). Therefore, the average delay between two keypresses is 500 ms, which is our defined interval between keypresses.
2) *Character set*: In our experiments, we use the printable characters (specified on the phone's QWERTY keyboard) only. In addition, we define the character set to compose of only characters that require one keypress. For example, a capital letter "A" requires two keypresses: ALT + A. We want to prevent an experimental scenario where the message length in memory is inconsistent with the total number of keys pressed.
3) *Message length*: We consider 3 different message lengths for the experiments. In each message, it includes one keypress for the final ENTER key. The 3 message lengths (in characters) are:
   - 75 – We assume that the user types at least one sentence (which is meaningful and of forensic value) in the message. The suggested English sentence length was 15 words (AskOxford). With an average of 5 characters per word, we have 75 characters in each sentence.
   - 150 – We assume that a short text message is 150 characters. This is obtained from computing the average of the upper limits of an SMS (160 characters) and a Twitter message (140 characters).
   - 225 – We observe that the above parameters can also emulate 1 or 2 sentences. Here, we look at the situation where the user types a longer message and assuming that it composes of 3 sentences on average.

We do realise that the message lengths in IM applications are dependent on other factors such as the topic being discussed and the individual user's style of chatting. These factors are hard to define and we only propose the above lengths as estimated indicators to perform the investigations.

4) *Message sending interval*: The interval between the sending of the messages depends on the interval between keypresses, k, and the message length, l. Assuming negligible reading time and network delay, we define 2 experiment scenarios with regards to the message sending interval. In both scenarios, the experiment always starts with the phone initiating the message typing and sending to the PC.

In the first scenario (refer to Fig. 2, where no waiting time is considered), once the first message has been "typed" and sent by the phone (i.e. the MSG and Monkey), the phone starts typing and sending the next message at the same time that the PC (i.e. the Chat Bot) starts typing and sending its first message. This is the worst case scenario as there is the continuous sending and receiving of messages and emulates the situation where the user does not wait for a reply before typing (and sending) a new message.

In the second scenario (refer to Fig. 3, where waiting time is included), once the first message has been "typed" and sent by the phone, the PC starts typing its first message while the phone waits. After the PC has finished typing and sent the message, the phone proceeds to type and send its next message. This scenario emulates the situation where both users take turn to send and wait for a response from the other.

The message sending interval in each direction is therefore $(k(l-1))$ for the no-wait scenario and $(2k(l-1))$ for the waiting scenario.

The parameters discussed above are defined in the MSG (and processed by Monkey) on the phone, and the Chat Bot on the PC.

4.4. *Evidence persistency examination*

Firstly, we conducted a simple experiment to study the persistency of the chat messages in the volatile memory. The message length was set to 75 characters and the interval between the keypresses to 500 ms. The experiment was similar to the wait scenario. We started the experiment with the phone typing and sending the first message. After which, a memory

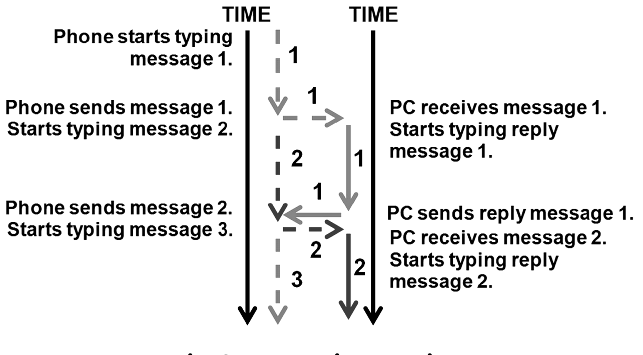
TIME
TIME
Phone starts typing message 1.
Phone sends message 1. Starts typing message 2.
Phone sends message 2. Starts typing message 3.
PC receives message 1. Starts typing reply message 1.
PC sends reply message 1.
PC receives message 2. Starts typing reply message 2.
1
2
3

Fig. 2 – No-wait scenario.

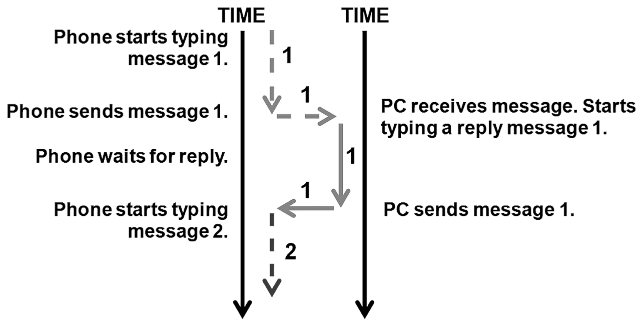
TIME
TIME
Phone starts typing message 1.
Phone sends message 1.
Phone waits for reply.
Phone starts typing message 2.
PC receives message. Starts typing a reply message 1.
PC sends message 1.
1
2

Fig. 3 – Waiting scenario.

dump was performed immediately. The PC then proceeded with typing and sending its first message. A memory dump was also performed immediately after the message was received. Therefore, the memory dump interval was the same as the typing time for a message, which was 37 s. A total of 30 messages were typed and sent (i.e. 15 in each direction).

Figs. 4 and 5 show the persistency of the outgoing messages (from phone) and incoming messages (to phone), respectively. The dumps with the odd IDs were performed after each outgoing message was sent, while the dumps with even IDs were performed after each incoming message was received.

A vertically plotted line at a dump ID gives an idea of the persistency of past messages in a particular dump, while a horizontally plotted line at a message ID gives an idea of the persistency of the message in subsequent dumps. The plotted parallel diagonal lines show the persistency of each message in the subsequent dumps.

For the outgoing messages, the message with ID 9 had the highest persistency which lasted for 8 intervals (i.e. 4.9 min). 100% and 92.86% of the outgoing messages persisted for an interval (i.e. 37 s) and two intervals, respectively. The persistency of each message diminished gradually in the subsequent dumps.

For the incoming messages, the message with ID 4 had the highest persistency, lasting for 6 intervals (i.e. 3.7 min). For the incoming messages, 64.29% and 50% persisted for an interval and two intervals, respectively. The persistency of the messages diminished quickly after that.

We observed from the results that the outgoing messages had a higher persistency than the incoming messages.

4.5. *Memory dump interval investigation*

In this part of the experiments, we investigated the effect of the different memory dump intervals in different chat scenarios. Some of the main parameters (i.e. keypress interval, character set, message lengths and message sending interval) for these experiments were defined at the beginning of this section (i.e. Section 4.3). In the experiments, a total of 30 messages were typed and sent, with 15 in each direction. The memory dumping on the phone was not performed immediately after each message was sent or received. Instead, we defined a finite dump interval for each experiment. One important consideration was that a larger dump interval is always desirable since

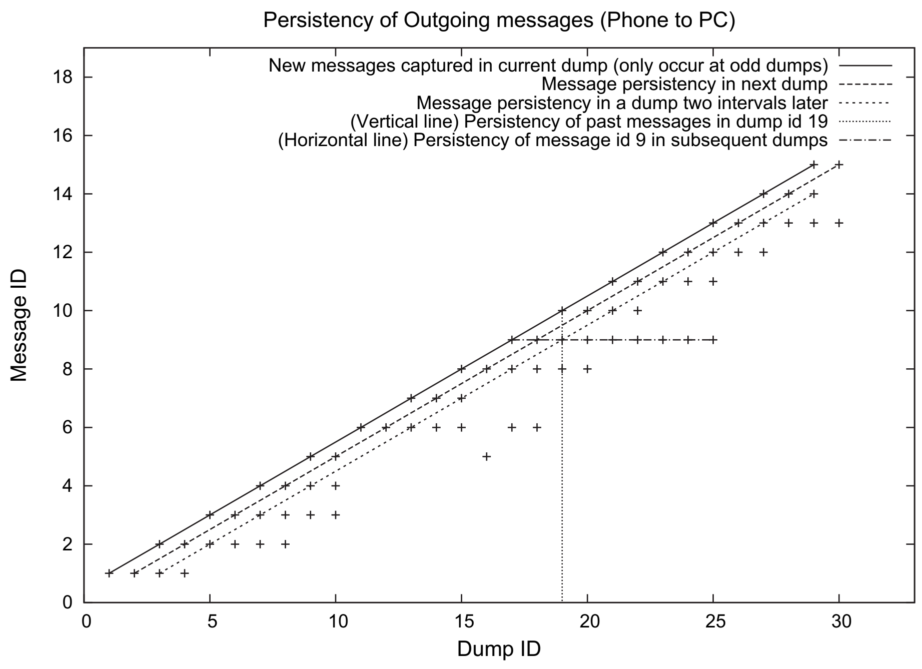

Fig. 4 – Persistency of outgoing messages.

fewer dumps consume lesser of the phone's valuable and limited resources (e.g. processing power, battery).

For the waiting scenario, we conducted 2 sets of experiments with the dump intervals of 40 and 60 s. The results are presented in Table 1. We observed that 100% of the evidence can be acquired successfully in the case of the outgoing messages even with the long dump intervals. For the incoming messages, the average evidence acquisition rate was 97.8% with a dump interval of 40 s and 95.6% with a dump interval of 60 s.

For the no-waiting time scenario, as the outgoing and incoming messages were being sent and received continuously, we reduced the dump interval to enable a more reliable evidence capturing. We conducted 4 set of experiments in this scenario, with each having a memory dump interval of 5, 10, 20 and 30 s. The results are presented in Tables 2 and 3.

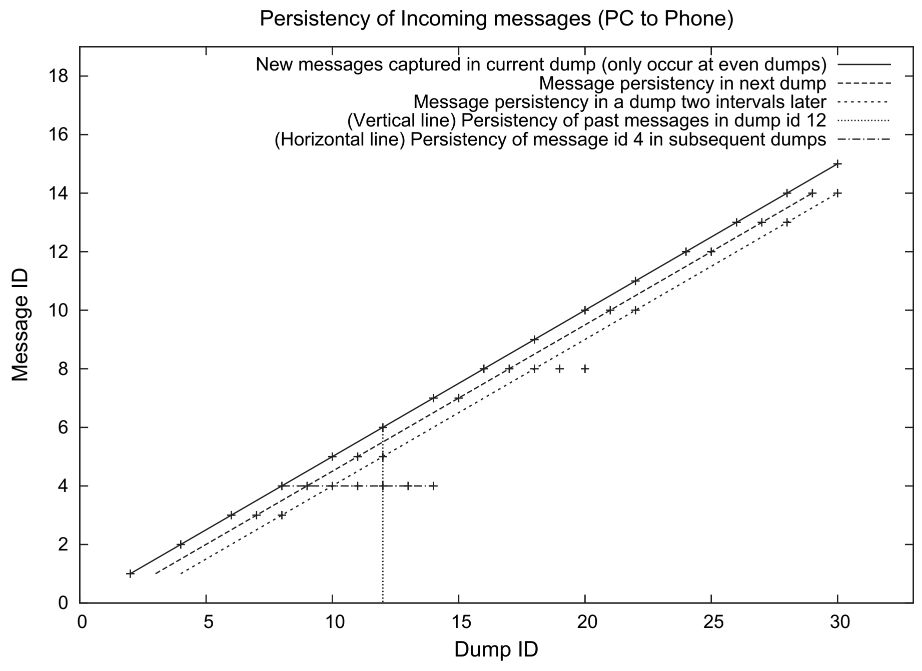

Fig. 5 – Persistency of incoming messages.

Table 1 – Captured (whole) messages in waiting scenario with dump intervals of 40 and 60 s.

Message length (Chars)	Dump interval			
	40 s		60 s	
	Outgoing Msgs	Incoming Msgs	Outgoing Msgs	Incoming Msgs
75	15/15	15/15	15/15	13/15
150	15/15	15/15	15/15	15/15
225	15/15	14/15	15/15	15/15

Table 2 – Captured (whole) messages in no-wait scenario with dump intervals of 5 and 10 s.

Message length (Chars)	Dump interval			
	5 s		10 s	
	Outgoing Msgs	Incoming Msgs	Outgoing Msgs	Incoming Msgs
75	15/15	15/15	15/15	13/15
150	15/15	15/15	15/15	13/15
225	15/15	15/15	15/15	13/15

Table 3 – Captured (whole) messages in no-wait scenario with dump intervals of 20 and 30 s.

Message length (Chars)	Dump interval			
	20 s		30 s	
	Outgoing Msgs	Incoming Msgs	Outgoing Msgs	Incoming Msgs
75	15/15	11/15	15/15	12/15
150	15/15	13/15	15/15	13/15
225	15/15	10/15	15/15	13/15

We observed that 100% of the evidence can be acquired successfully in the case of the outgoing messages for all the dump intervals. When the dump interval was 5 s, all the incoming messages were successfully captured as well. However, as the interval increased, the rate of successful acquisition dropped for the incoming messages (e.g. 86.7%, 75.6% and 84.4% for dump intervals of 10, 20 and 30 s, respectively).

Based on the experiments, we can conclude that the outgoing messages had a higher persistency than the incoming ones. We also noticed during our experiments that due to the need for internal processing of the outgoing messages to be sent out through the network, multiple copies in different formats can exist in the memory. In contrast, the incoming messages only existed in at most two copies in the memory. Therefore, the persistency for the incoming messages was lower and the successful acquisition rate was also lower than that for the outgoing messages.

5. Future work

We are designing an optimized evidence acquisition tool which utilises our analysis results. The tool will be used to study the acquisition of evidence in an actual communication scenario. We will also be investigating on porting the system to other mobile phone platforms.

6. Conclusions

In this paper, we identified the need for a live memory forensic analysis for mobile phones due to the difficulties in recovering evidence residing in the volatile memory. We proposed an automated system that analyses the dynamic properties of the mobile phone's volatile memory and carries out real-time evidence acquisition. The system supported the analysis of the persistency of evidence generated by interactive and communication based applications. Different communication scenarios with varying parameters (e.g. message lengths, messaging intervals, dump intervals, keypresses interval) were investigated. Our experiment results showed that the outgoing messages (from the phone) had a higher persistency than the incoming messages (to the phone). With varying message lengths, we consistently achieved a 100% evidence acquisition rate with the outgoing messages in all scenarios. For the incoming messages, the acquisition rates were 95.6% and 97.8% for dump intervals of 40 and 60 s, respectively, in the scenario where each party waited for the other before responding with a reply message. In the scenario where both parties rapidly continued to send messages without waiting for a reply, the acquisition rates were 100%, 86.7%, 75.6% and 84.4% for dump intervals of 5, 10, 20 and 30 s, respectively. Therefore, we can conclude that in a more realistic scenario where the parties occasionally take turns to send messages and consecutively send a few messages, the evidence acquisition rate is acceptable to capture most of the data to allow a more detailed forensic investigation.

REFERENCES

Adelstein F. Live forensics: diagnosing your system without killing it first. Communications of the ACM February 2006;49 (2):636–66.

Ahmed R, Dharaskar RV. Mobile forensics: an overview, tools, future trends and challenges from law enforcement perspective. In: 6th International Conference on E-Governance, ICEG, Emerging Technologies in E-Government, M-Government, December 2008 pp. 312–23.

Al-Zarouni M. Introduction to mobile phone flasher devices and considerations for their use in mobile phone forensics. In: Proceedings of the 5th Australian digital forensics conference; December 2007.

Ashcroft J, Daniels DJ, Hart SV. Forensic examination of digital evidence: a guide for law enforcement. National Institute of Justice (NIJ); April 2004. Special Report.

AskOxford. Better writing, http://www.askoxford.com/.

Ayers R, Jansen W, Moenner L, Delaitre A. Cell phone forensic tools: an overview and analysis update. National Institute of Standards and Technology; March 2007. Technical Report 7387.

Berte R, Dellutri F, Grillo A, Lentini A, Me G, Ottaviani V. Fast smartphones forensic analysis results through mobile internal acquisition tool and forensic farm. International Journal of Electronic Security and Digital Forensics March 2009;2(1): 18–28.

Bhadsavle N, Wang JA. Validating tools for cell phone forensics. Southern Polytechnic State University; 2008. Technical Report CISE-CSE-08-05.

Carrier BD, Grand J. A hardware-based memory acquisition procedure for digital investigations. Digital Investigation February 2004;1(1):50–60.

Casadei F, Savoldi A, Gubian P. Forensics and SIM cards: an overview. International Journal of Digital Evidence Fall 2006;1(1):1–21.

Chen J. An introduction to android. Google I/O; 2008.

Distefano A, Me G. An overall assessment of mobile internal acquisition tool. In: Proceedings of the 8th Digital Forensics Research Conference (DFRWS), digital investigation. vol. 5, no. 1; September 2008, pp. S121–7.

Farmer D, Venema W. The coroner's toolkit; August 1999.

Forensic analysis of mobile phone internal memory. Advances in digital forensics, IFIP International Federation for information processing, vol. 194. Springer; March 2006. 191–204.

Hoog A, Gaffaney K. iPhone forensics. via Forensics Whitepaper; June 2009.

Hoog A. Android forensics, presented at Mobile Forensics World 2009; May 2009.

Husain MI, Sridhar R. iForensics: forensic analysis of instant messaging on smart phones. In: Digital forensics and cyber crime, vol. 31; January 2010. pp. 9–18.

Jansen W, Ayers R. Forensic software tools for cell phone subscriber identity modules. In: Conference on Digital Forensics, Association of Digital Forensics, Security, and Law (ADFSL); April 2006.

Jansen W, Delaitre A, Moenner L. Overcoming impediments to cell phone forensics. In: Proceedings of the 41st Hawaii International Conference on system sciences; 2008.

Kiley M, Dankner S, Rogers M. Forensic analysis of volatile instant messaging. In: Advances in digital forensics IV, IFIP International Federation for information processing, vol. 285. Springer; August 2008. pp. 129–38.

Kim K, Hong D, Chung K, Ryou JC. Data acquisition from cell phone using logical approach. In: Proceedings of world academy of science, engineering and technology, vol. 26; December 2007.

Kumparak G. Google: android now shipping on 60,000 handsets per day, http://www.mobilecrunch.com; February 2010.

Mokhonoana PM, Olivier MS. Acquisition of a Symbian smart phone's content with an on-phone forensic tool. Department of Computer Science, University of Pretoria; 2007.

Petroni Jr NL, Walters A, Fraser T, Arbaugh WA. FATKit: a framework for the extraction and analysis of digital forensic data from volatile system memory. Digital Investigation December 2006;3(4):197–210.

Schatz B. BodySnatcher: towards reliable volatile memory acquisition by software. In: Proceedings of the 7th Digital Forensics Research Conference (DFRWS), Digital Investigation, vol. 4, no. 1; September 2007, pp. S126–34.

Sears A, Revis D, Swatski J, Crittenden R, Shneiderman B. Investigating touchscreen typing: the effect of keyboard size on typing speed. University of Maryland; 1992. Technical Report CS-TR-2662.

Simon M, Slay J. Enhancement of forensic computing investigations through memory forensic techniques. In: International Conference on availability, reliability and security, 2009, p. 995–1000.

Willassen S. Forensics and the GSM mobile telephone system. International Journal of Digital Evidence Spring 2003;2(1):1–17.

Williamson B, Apeldoorn P, Cheam B, McDonald M. Forensic analysis of the contents of Nokia mobile phones. In: Proceedings of the 4th Australian digital forensics conference; December 2006.

Vrizlynn Thing received the Ph.D. degree in Computing from the Imperial College London, U.K., for her work on detecting and mitigating Distributed Denial-of-Service attacks, and the B.Eng. (Hons) and M.Eng. degrees in Electrical and Electronics Engineering from the Nanyang Technological University, Singapore. She currently leads the Digital Forensics research group at the Institute for Infocomm Research, Singapore. Her research interests include digital forensics, system and network security, networking protocols, and optical fiber communications.

Kian-Yong Ng is a Computer Science student, specialising in Information Security, at the School of Computing, National University of Singapore. His research interests include system and network security.

Ee-Chien Chang received the BSc and MSc degree from the National University of Singapore (NUS), in 1991 and 1993 respectively and the PhD degree from New York University in 1998. In 1999, he was a Postdoctoral Fellow with the Center for Discrete Mathematics and Theoretical Computer Science (DIMACS). He was with the Department of Computational Science, NUS for two years. Currently, he is an Associate Professor in the School of Computing, NUS. His research interests include multimedia security, and information security.