Extraction Of HTTP Messages For Retrieval Of Live Evidence

Ipsita Mohanty, R. Leela Velusamy
Department of Computer Science and Engineering
National Institute of Technology
Tiruchirappalli, India
ipsita.mohanty689@gmail.com, leela@nitt.edu

***Abstract*—There has been a surge towards physical memory forensics with recent development of valuable tools and techniques in acquisition and analysis. However, most of the research is skewed towards the Operating System architecture with primary focus on processes, threads, kernel modules etc. There has been little research towards retrieval of application level data. Moreover, the research is majorly application specific and keyword search oriented, involving text string search tools which use match and/or indexing algorithms to search digital evidence. Our paper aims to unearth application level data through a generic approach and provide some degree of automation to reduce the string search dependency. The paper revolves around the web browser and the HTTP protocol which are the basic elements of the interface between a user and the internet. Programs have been developed for the extraction of HTTP message headers from the acquired RAM image and a detailed analysis of how the information in these headers is crucial, follows.**

Keywords—Digital Forensic; Live Evidence; Application level Data; HTTP message

I. INTRODUCTION

The internet has brought about a revolution in the way we lead our life; the manner in which we work, socialize, gather and share information. One-third of the world's population is online, an increase of 528 percent over the past 10 years [1]. Activities such as social-networking, chatting, online shopping, and online banking have become common among people on account of convenience and simplicity. As a result, the internet has become a source of plethora of information and has caught the attention of criminals. Cyber crimes such as hacking, data theft, spreading virus/worms, identity theft, and e-mail spoofing have become widespread.

Any cyber crime would inevitably involve the use of internet and hence, the criminal is also an internet user who can leave behind potential information. The most common interface between a user and the internet is the web browser. The memory used and files stored by the web browser are sources of important information. The web browser acts as the client of the user, converts the input information into HTTP requests and sends to the target server which services requests by sending back HTTP response messages. HTTP messages are accompanied by various headers and the message body. HTTP messages contain information such as entered search words, user-id and password, the URL of the websites accessed, the files downloaded, etc. The extraction of such fields can help in profiling the target user and hence aid in an investigation process. RAM is a storage media from which such information can be extracted.

RAM being the first point of information storage, though volatile, the plethora of information available has caught attention of researchers. Over the recent years, there has been increased focus on the need to develop tools and techniques to capture and analyze Physical Memory content. Though there has been quite a bit of research on acquisition and analysis of physical memory, the direction has been generic with focus primarily on the operating system architecture. There has been significant improvement in automation of image acquisition techniques but the analysis of the acquired images is usually manual, case-dependent, and relies on investigator expertise. In addition, memory analysis techniques are limited to searching predefined strings or series of bytes [2].

There is a need for a structured method; beyond processes, threads and predefined keyword/fingerprint searches. Our paper aims to shift the focus from the conventional approaches of memory analysis and keyword dependent search. The contribution of our paper is twofold: 1. Focus on recovery of application level data (browser and internet applications) from the RAM image 2. Analysis of physical memory image through automated extraction of HTTP message headers

The organization of the remaining portion of the paper is as follows. Section II provides a brief review of prior work and the motivation for our attempt. Section III gives a description about the HTTP protocol and Section IV covers the methodology adopted. Section V enlists the performance analysis of the algorithm developed for extraction. The paper concludes with discussion about the future scope in Section VI.

II. BACKGROUND

With cyber crime on the rise, Digital forensic has become an important part of Forensic Science. Digital forensic involves acquisition of data from different digital sources, analysis of the data, extraction of evidence, preservation and presentation

ISBN:978-1-4799-1024-3/13/$31.00 ©2013 IEEE

of the evidence [3]. The sources of acquisition include digital media spotted in the crime area such as CDs, DVDs, flash drives, floppy disks, memory cards, mobile phones, network devices, RAM, etc. These media are classified into two types: static and live (volatile), on the basis of the persistency of data within them. Acquisition and analysis of data from static devices (CDs, DVDs, hard disks etc.), commonly referred to as static data, have undergone rigorous research and plenty of tools and techniques have been developed in this area. Research has been conducted in network packet capture and analysis too. However, live acquisition has stayed out of the limelight. Digital forensic investigators may be able to gather evidence from a system while it is in its running or 'live' state. Acquisition of live evidence is also possible if the system has been hibernated, since the hiberfil.sys, stored in the local disk, contains the contents of the RAM. The information which can be retrieved includes details about network connectivity, running processes, loaded DLLs and files, user names, passwords, encryption keys for applications, etc. Given the profound scope and high utility, acquisition and analysis of live data has gained importance lately.

Acquisition of memory images has been in focus with noticeable research in this area. Schatz developed Body-Snatcher which injects an independent acquisition OS to take snapshots of the host OS memory [4]. Halderman et al. proposed memory acquisition via cold booting [5]. Carrier and Grand proposed a hardware-based solution called "Tribble" wherein a PCI expansion card is used to capture the memory content through Direct Memory Access [6]. Notable software-based solutions include Data Dumper, Memory DD, Widows Memory Toolkit, Memoryze, WinEn, FTK Imager, KntDD and Nigilant32 [7].

There exist some tools for analysis of acquired physical memory images. But the research direction is mostly skewed towards operating system architecture with the focus being on processes, threads and kernels. For example, PTFinder, a signature-based scanner, developed by Schuster reveals hidden and terminated processes and threads [8]. MemParser, a tool developed by Betz, reconstructs processes lists and extracts process specific information [9]. KnTList, a command line tool, by Garner, rebuilds the virtual address space of the system process and other processes. Volatility Framework and Memoryze are some other tools which also have OS architecture focus and help list down processes, kernel modules, loaded files and DLL modules, drivers, open connections, etc. [2].

Zhao and Cao made an attempt to move away from conventional OS architecture and recovered application level information such as user IDs and passwords from the memory using hiberfil.sys, Windows crash dump file, pagefile and direct memory access [10]. Mike Dickson found out remnants of conversations in each of AIM, MSN and Yahoo messengers through analysis of RAM contents [11, 12, 13]. Simon and Slay were able to retrieve communication content, communication history, contacts, passwords, and encryption keys for Skype [14]. In a recently published paper, Mohanty and Velusamy were able to extend the work of Simon and Slay with focus on application level data of a browsing session by retrieving user-ids and passwords for internet applications such as Facebook, Gmail, etc. [15]. However, these attempts involved manual searching for keywords/strings/series of bytes which needed to be decided apriori and differed for different applications. This manual searching is more of a hit and trial method and may not achieve the desired result. In this era of cheap and vast storage devices with the diversity of applications available, the investigator really needs to know what he is searching for, where it is available and how it can be extracted in an efficient manner [16]. Digital investigators must differentiate between the relevant and irrelevant data (junk) since different crimes result in different types of digital evidence. Searching through a huge volume of data manually leaves the scope for missing out critical information and hence renders the above researches slightly inefficient. There is need for generalized and automated methods for the application-layer evidence analysis of huge amount of data acquired from RAM.

A shift from OS architecture approach and the need for a generic analysis methodology with automation is what our paper caters to. The focus of our paper is HTTP, the most commonly used application layer protocol for transfer of messages. A user in a browsing session sends and receives numerous messages through HTTP. Each HTTP message contains crucial information which is tapped in the RAM image. This paper discusses a method to extract information in the form of message headers from acquired RAM images with an automated program, which further facilitates extraction of information from the message body.

III. HTTP PROTOCOL

HTTP is the most used application level request/response protocol between web browsers and web servers. A Message is the basic unit of HTTP communication. A client e.g. the web browser sends a request to the server in the form of a request method, URI, and protocol version, followed by a MIME-like message containing request modifiers, client information, and possible body content over a connection with a server. The server responds with a status line, including the protocol version and a success or error code, followed by a MIME-like message containing server information, entity Meta information, and possible entity-body content [17]. So the contents entered by the user in a web browser interface e.g. URLs, user-ids, passwords, search words, etc. are stored in the HTTP request messages and from the other side, the response contents e.g. user pictures, messages, etc. are stored in the HTTP response messages. This clearly indicates that the request and response messages are sources of crucial information and can be targeted to retrieve significant details.

An HTTP Message consists of a structured sequence of octets. Request and Response messages use the generic message format of RFC 822 for transferring entities (the payload of the message) [18]. Both types of messages consist of a start-line, zero or more header fields (also known as "headers"), an empty line (i.e., a line with nothing preceding the CRLF) indicating the end of the header fields, and possibly a message-body, as shown in Fig. 1 [17].

The start-line is a Request-line in case of request message and a Status-line for response message. The formats of Request-line and Status-line are as given in Fig. 2.

Each message header field consists of a name followed by a colon (":") and the field value. The field value MAY be preceded by any amount of Linear White Space (LWS), though a single space (SP) is preferred. The RFC 2616 can be referred for details about the terms used in HTTP messages [17].

```
HTTP-message = Request | Response
             = start-line
             *(message-header CRLF)
             CRLF
             [Message-body]

start-line = Request-Line | Status-Line
```

Figure 1. HTTP message format

```
Request-Line=Method SP Request-URI SP HTTP-Version CRLF
        Where
          SP: space
          Method="OPTIONS" | "GET" | "HEAD" |
                 "POST" | "PUT" | "DELETE" |
                 "TRACE" | "CONNECT" | extension-
                 method
         Request-URI ="*"|absoluteURI|abs_path|authority

Status-Line=HTTP-Version SP Status-Code SP Reason-Phrase
        CRLF
```

Figure 2. Format of request line and status line

IV. METHODOLOGY

In our approach, we have designed regular expressions to extract the HTTP message headers through parsing. HTTP message headers have a particular structure which remains constant across all messages transmitted. This trend has been leveraged by developing a regular expression specifically to extract the message headers. The message body was retrieved using the information present in various fields of the headers.

A. Acquisition of RAM Image

A browsing session was started and access to internet was obtained by logging into Sonicwall (a firewall interface). The application Facebook was opened in the browser and logged into. Using Nigilant32, an image of the RAM was taken. The choice of the application was based on popularity, frequency of usage and importance of contained data.

The system being used for acquiring RAM image was a Lenovo 0768 HBQ laptop with following specifications:

- OS: Windows XP Professional, Service Pack 2
- Processor: Intel Pentium Dual-Core Processor T2080 @ 1.73GHz 794MHz
- Physical Memory: 512 MB
- Hard Disk: 80 GB
- Page file size: 0MB
- Internet browser: Mozilla Firefox 6.0
- Web application: Facebook

The page file size was set to zero. This was done to ensure that all contents are available in the RAM with nothing being swapped out to the virtual memory.

B. Message Header Extraction

For extraction of message headers, regular expressions were used and the program was written in Python 2.7. As per the HTTP message format described in Section III, the rules for the set of possible strings to be matched were specified.

The regular expressions framed were as given in Fig. 3. (?:GET|POST) is for matching the method type. Only these two methods have been taken into consideration because of presence in most of the http messages. '.' is used to match any character except a newline character. The meta-characters '*', '+', {n} specify that the previous character is to be repeated a certain number of times: '*' - zero or more times, '+' - one or more times, '{n}' - exactly n times. Combining both, '.*' is used to match a sequence of multiple characters. HTTP\/\d+\.\d+ is used to indicate the HTTP version such as HTTP/1.1 or HTTP/1.0. The meta-characters '/' and '.' are preceded by a backslash character to ensure that they are searchable as a part of the string. '\d' matches any decimal digit. [\w-] matches any alphanumeric character and the special character '-'. \xh matches a character with hexadecimal value h; hence, \x0D, \x0A for carriage return, line feed respectively and [\x20-\xFF] for characters with ASCII hex value from 20 to FF [19].

```
request_line  = (?:GET|POST).*HTTP\/\d+\.\d+\x0D\x0A

status_line = HTTP\/\d+\.\d+\x20\d{3}\x20.*\x0D\x0A

msg_header = [\w-]*:[\x20-\xFF]*\x0D\x0A
```

Figure 3. Regular expressions

The message headers were extracted through automated programs following the algorithm given below, involving the regular expressions constructed. Separate programs were written for request and response messages. The image of RAM was provided as input to these programs and outputs were text files containing the request or status line along with HTTP message headers.

Algorithm:

Input: An image file of RAM

Output: A text file containing the request or status line and the message headers

Steps:

1. Open the input file in binary mode
2. Find all strings matched by the regular expression for request/status line and save them in a list
3. **if** length (list)>0
4. **then for** x **in** list
5. **do while** 1
6. **do** read a single line from the file

```
7.      if line is NULL
8.        then print 'file over'
9.        break
10.     else if x is present in line
11.       then print x
12.       while 1
13.         do read a new line
14.         if the regular expression for message header
        matches the line
15.           then print the matched message header
16.         else
17.           if the regular expression for request/status line
        matches the line
18.             then print the matched request/status line
19.             Continue
20.           else
21.             break
22.       print 'headers printed'
23.       print a blank line
24.       break
```

C. Message Body Extraction

The message headers, after retrieval, were analyzed for the presence of message body. If present, the message body was extracted which contains information relevant to target user. The presence of a message-body in a request message depends upon the request method and is signaled by the inclusion of a Content-Length or Transfer-Encoding header field in the request's message-headers. For response messages, the inclusion of a message-body is dependent on both the request method and the response status code. The length of message body in number of octets is determined by the Transfer-Encoding header field or Content-Length header field. If a message contains both Transfer-Encoding header field and Content-Length header field, the latter MUST be ignored [17].

V. Performance Analysis

This section covers the performance analysis of the two programs written separately for extraction of HTTP request and response messages' headers. The analysis brings out the utility of the extracted headers in terms of critical information retrievable. Examples have been depicted for each category of messages to showcase how the message headers have enabled methodical extraction of information. The content-type and content-length fields are of primary concern for the extraction of the message body associated with the message headers. The message body revealed critical information for the multimedia rich application facebook. Provided below is a detailed analysis of the message headers extracted.

A. HTTP Request Message

The output text file for HTTP request messages had contents as shown in the example in Fig. 4. A detailed explanation of the meaning of each line with the number in the bracket referring to the decimal offset address of the line in the RAM image is given below.

The first line being the request line indicates that it is a REQUEST message and is followed by the message headers. Since the request method used was POST, there is a probability of the existence of a message body. However, the Content-Length field (21129798) value was 0 indicating that the message body was empty. The Content-Length entity-header field indicates the size of the entity-body, in decimal number of OCTETs. Given a non-zero content-length value, a message body can contain information such as user-id, password, session-id, etc. The host name (21129303) and request URI were combined to find out that the webpage requested was 192.168.20.1/ usrHeartbeat.cgi. The user-agent or web browser used, was Mozilla Firefox 6.0 (21129323). The request-headers: Accept, Accept-Language, Accept-Encoding and Accept-Charset specify the acceptable features of the response content requested. The connection field (21129586) had value "keep-alive" which means a persistent connection was being used for the session. Once the connection is established between the web browser and web server, it can be used for all HTTP request and response messages communicated in that session.

```
POST /usrHeartbeat.cgi HTTP/1.1
21129303:Host: 192.168.20.1
21129323:User-Agent: Mozilla/5.0 (Windows NT 5.1; rv:6.0)
        Gecko/20100101 Firefox/6.0
21129400:Accept:
        text/html,application/xhtml+xml,application/xml;q=0.9,*/
        *;q=0.8
21129473:Accept-Language: en-us,en;q=0.5
21129506:Accept-Encoding: gzip, deflate
21129538:Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
21129586:Connection: keep-alive
21129610:Referer:
        http://192.168.20.1/loginStatus.html?1stLoad=yes
21129610:Referer:
http://192.168.20.1/loginStatus.html?1stLoad=yes
21129669:Cookie: temp=temp; SessId=523518834;
        PageSeed=7e88fbfc81a9a474bc1fc4c9cce95f1f
21129749:Content-Type: application/x-www-form-urlencoded
21129798:Content-Length: 0
```

Figure 4. Extracted request line and headers of an HTTP request message

Examples of the headers of request messages extracted from the RAM image taken, while facebook was logged into, are shown in Fig. 5 and 6. In this case, the request method is 'GET' and 'Content-Type' and 'Content-Length' fields are absent. So, it can be concluded that no message body was included with these HTTP request messages.

```
GET /hprofile-ak-ash2/275881_100001413808319_569676_q.jpg
HTTP/1.1
21308100:Host: profile.ak.fbcdn.net
21308128:User-Agent: Mozilla/5.0 (Windows NT 5.1; rv:6.0)
        Gecko/20100101 Firefox/6.0
21308205:Accept: image/png,image/*;q=0.8,*/*;q=0.5
21308248:Accept-Language: en-us,en;q=0.5
21308281:Accept-Encoding: gzip, deflate
21308313:Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
21308361:Connection: keep-alive
21308385:Referer: http://www.facebook.com/
```

Figure 5. A request message for the domain facebook

In case of Fig. 5, the message header (21308205) shows that there was a request sent by the client browser for a picture of the user having facebook id 100001413808319 from the host 'profile.ak.fbcdn.net' (21308100). Once this id is known, the public information about the corresponding user can be retrieved from facebook's Graph API, by entering https://graph.facebook.com/id into a browser's address (URL) box.

The Graph API is an application programming interface which presents a simple, consistent view of the Facebook's social graph, uniformly representing the objects in the graph and the connections between them. Every piece of data stored in Facebook's database has a unique ID associated with it, which is usually a label or a number [20]. In our example, 100001413808319 is the ID (of the personal node) associated with the user's public information. When we enter the URL https://graph.facebook.com/100001413808319 into a browser's address (URL) box, an API call is made to the Graph API and Facebook returns a javascript (.js) file containing the publically available information.

Using the ID derived from the example above, when we enter the URL https://graph.facebook.com/100001413808319, the information obtained is:


```
{
  "id": "100001413808319",
  "name": "Deepa N Sarma",
  "first_name": "Deepa N",
  "last_name": "Sarma",
  "link": "http://www.facebook.com/deepan.sarma",
  "username": "deepan.sarma",
  "gender": "female",
  "locale": "en_US"
}
```


```
GET
/x/1485986247/4163080690/true/p_100001759385636=0
HTTP/1.1
74864712:Host: 0.186.channel.facebook.com
74864746:User-Agent: Mozilla/5.0 (Windows NT 5.1;
        rv:6.0) Gecko/20100101 Firefox/6.0
74864823:Accept:
        text/html,application/xhtml+xml,application/xml
        ;q=0.9,*/*;q=0.8
74864896:Accept-Language: en-us,en;q=0.5
74864929:Accept-Encoding: gzip, deflate
74864961:Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
74865009:Connection: keep-alive
74865033:Referer:
        http://0.186.channel.facebook.com/iframe/11?r=htt
        p%3A%2F%2Fstatic.ak.fbcdn.net%2Frsrc.php%2
        Fv1%2FyN%2Fr%2FgK0PYC9f6km.js&r=http%
        3A%2F%2Fstatic.ak.fbcdn.net%2Frsrc.php%2Fv1
        %2Fyr%2Fr%2FTPa-
        xHTGNyJ.js&r=http%3A%2F%2Fstatic.ak.fbcdn.
        net%2Frsrc.php%2Fv1%2Fyw%2Fr%2FBKYe49
        wUgWE.js&r=http%3A%2F%2Fstatic.ak.fbcdn.ne
        t%2Frsrc.php%2Fv1%2FyZ%2Fr%2F0OPyWwm
        4qNm.js&r=http%3A%2F%2Fstatic.ak.fbcdn.net
        %2Frsrc.php%2Fv1%2FyE%2Fr%2FSp2IUK7A8
        Z2.js
74865472:Cookie: datr=URxWTtwBwwh1_54FeaDLy-gI;
        act=1314266194344%2F3; L=2;
        c_user=100001759385636; lu=RARCkXj2hs0G-
        W5Fc6b4b79g; sct=1314266226;
        xs=60%3A6125804b6c38989b5fd79605c7d01286
        ;
        presence=EM314266229L186REp_5f1B0175938
        5636F0X314266228246K0H1U0OQ0EsF0CEblF
        DacF0EutF0PCC; p=186
```

Figure 6. Another request message for the domain facebook

In the request message shown in Fig. 6, the facebook ID of the target user can be retrieved from the 'c_user=100001759385636' field name: value pair stored in the Cookie header. As described in the previous example, using the Facebook Graph API, the public information about the target user as given below can be collected.


```
{
  "id": "100001759385636'",
  "name": "Ipsita Mohanty",
  "first_name": "Ipsita",
  "last_name": "Mohanty",
  "link": "http://www.facebook.com/ipsita.mohanty.94",
  "username": "ipsita.mohanty.94",
  "gender": "female",
  "locale": "en_US"
}
```


B. HTTP Response Message

An example of an HTTP response message is given in Fig. 7.

```
HTTP/1.1 200 OK
21156207:Content-Type: image/jpeg
21156233:Content-Length: 2541
21156255:X-Backend: hs203.ash4
21156278:X-Blockid: 276247
21156297:Last-Modified: Fri, 01 Jan 2010 00:00:00 GMT
21156343:Cache-Control: max-age=288893
21156374:Expires: Sun, 28 Aug 2011 18:12:05 GMT
21156414:Date: Thu, 25 Aug 2011 09:57:12 GMT
21156451:Connection: keep-alive
```

Figure 7. Extracted status line and headers of an HTTP response message

The content type is image/jpeg (21156207) and content-length 2541 (21156233). So, the message body, a jpeg image of 2541 bytes is loaded into memory. A jpeg image starts from the hex byte sequence FFD8FFE000104A464946 and ends with FFD9. In search for similar content, we selected the bytes starting from the end of the message headers and the empty line till the hex byte sequence FFD9 and saved in a jpeg file.

The length was found to be 2605 bytes. Considering the content length given in message header, a content of length 2541 bytes was selected instead of 2605 after the delimiting empty line and saved in another jpeg file, even though it did not contain the hex byte sequence FFD9. In both the cases, the image obtained in a picture viewer is as shown in Fig. 8. Some more images extracted through analysis of other HTTP response messages' headers are shown in Fig. 9.

Each image obtained is an amalgamation of multiple pictures. It is evident that multiple overwriting of images has occurred over the same addresses. Though the content length is 2541 bytes in this response message (Fig. 7), the sequence FFD9 was located after 2603 bytes, thereby indicating that it belongs to images loaded before the response message image. The image files found, though incomplete, can help in an investigation. Further analysis showed these were the profile pictures of the friends of the target user.

Figure 8. Image obtained from the HTTP response message

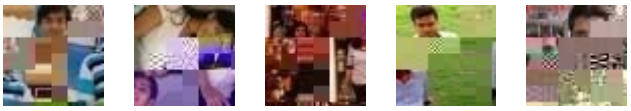

Figure 9. Images extracted through analysis of other HTTP response messages

VI. Conclusion And Future Work

The performance analysis clearly brings out the importance of an application level data approach and physical memory forensics. The HTTP messages have a well organized format and data structure which has been leveraged in establishing a generalized method for their extraction. The information retrieved from the HTTP messages is of high importance. The public information about the target user and his/her contacts were retrieved from the HTTP request messages. The HTTP response messages facilitated the extraction of image files. As a result, we were able to get an overview of the social profile of the target user which is critical to profiling and investigation.

Though we have attempted to automate the information retrieval, it is limited to extraction of HTTP message headers. Hence, after extracting the message headers through an automated method, message body was extracted and analyzed manually. As a part of future work, we would like to explore the possibility of automation of extraction of the message body contents. Moreover, our results are based on the most used application – Facebook. We would also like to explore if similar results can be obtained for multiple applications across multiple browsers.

References

[1] Internet World Stats. Retrieved from www.internetworld stats.com/stats.htm

[2] S. M. Hejazi, C. Talhi, and M. Debbabi, "Extraction of forensically sensitive information from windows physical memory," Digital Investigation, Vol. 6, Issue. Supplement, pp. 121-131, 2009.

[3] B. Carrier, "Defining Digital Forensic Examination and Analysis Tools," Digital Forensic Research Workshop (DFRWS), August 2002. Retrieved from http://www.dfrws.org/2002/papers/Papers/Brian_carrier.pdf

[4] B. Schatz, "BodySnatcher: Towards reliable volatile memory acquisition by software," Digital Investigation, Vol. 4, Issue. Supplement, pp. 126-134, 2007.

[5] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten, "LestWe Remember: Cold Boot Attacks on Encryption Keys," 17th USENIX Security Symposium (Sec '08), San Jose, CA, pp. 45-60, July 2008.

[6] B. D. Carrier and J. Grand, "A hardware-based memory acquisition procedure for digital investigations," Digital Investigation, Vol. 1, Issue. 1, pp. 50-60, 2004.

[7] S. Vömel and F. C. Freiling, "A survey of main memory acquisition and analysis techniques for the windows operating system," Digital Investigation, Vol. 8, Issue. 1, pp. 3-22, 2011.

[8] A. Schuster, "Searching for processes and threads in Microsoft Windows memory dumps," Digital Investigation, Vol. 3, Issue. Supplement, pp. 10-16, 2006.

[9] C. Betz, "Memparser analysis tool," Digital Forensic Research Workshop (DFRWS), 2005. Retrieved from http://www.dfrws.org/2005/challenge/memparser.shtml

[10] Q. Zhao and T. Cao, "Collecting sensitive information from windows physical memory," Journal of Computers, Vol. 4, Issue. 1, pp. 3-10, January 2009.

[11] M. Dickson, "An examination into AOL Instant Messenger 5.5 contact identification," Digital Investigation, Vol. 3, Issue. 4, pp. 227-237, 2006.

[12] M. Dickson, "An examination into MSN Messenger 7.5 contact identification," Digital Investigation, Vol. 3, Issue. 2, pp. 79-83, 2006.

[13] M. Dickson, "An examination into Yahoo Messenger 7.0 contact identification," Digital Investigation, Vol. 3, Issue. 3, pp. 159-165, 2006.

[14] M. Simon and J. Slay, "Recovery of Skype Application Activity Data From Physical Memory," Fifth International Conference on Availability, Reliability and Security (ARES 10), pp. 283-288, Feb 2010.

[15] I. Mohanty and R. L. Velusamy, "Recovery of Live Evidence from Internet Applications," in Advances in Computer Science, Engineering & Applications, vol. II, Advances in Intelligent Systems and Computing, Vol. 167, D. C. Wyld, J. Zizka and D. Nagamalai, Eds. Springer, 2012, pp. 823–834.

[16] J.R. Vacca, Computer forensic: computer crime scene investigation, Charles River Media, 2002.

[17] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1," RFC 2616, June 1999.

[18] D. H. Crocker, "Standard For The Format Of ARPA Internet Text Messages," RFC 822, August 13, 1982.

[19] Python v2.7.2 documentation. Retrieved from http://docs.python.org/release/2.7.2/

[20] Facebook Graph API. Retrieved from https://developers.facebook.com/docs/reference/api/