2011 IEEE International Conference on Privacy, Security, Risk, and Trust, and IEEE International Conference on Social Computing

Comparative Analysis of Volatile Memory Forensics

Live Response vs. Memory Imaging

Amer Aljaedi, Dale Lindskog, Pavol Zavarsky, Ron Ruhl, Fares Almari
Information Systems Security Management
Concordia University College of Alberta
Edmonton, Canada
{amer4554, fares.almari}@gmail.com, {dale.lindskog, pavol.zavarsky, ron.ruhl}@concordia.ab.ca

***Abstract*—Traditionally, incident responders and digital forensic examiners have predominantly relied on live response for volatile data acquisition. While this approach is popular, memory capacity has rapidly changed, making memory a valuable resource for digital investigation, by revealing not only running tasks, but also terminated and cached processes. This research presents the impact and the limitations of the conventional volatile forensic method, live response, in comparison to the alternative method, memory image analysis. The experiment's results demonstrate and we discuss the forensic effects of executing a live response toolkit, which alters the volatile data environment significantly in some cases and can overwrite potential evidence. Memory image analysis is also leveraged as an alternative approach that helps mitigate the risk of losing volatile evidence such as terminated and cashed processes, which are ignored during live response. This comparative analysis calls attention the capabilities of both methods in retrieving and recovering volatile data.**

Keywords- volatile data forensics; live response; memory analysis; incident response

I. INTRODUCTION

Volatile memory analysis has become a significant part of the digital investigation because there is digital evidence that resides only in physical memory (RAM) and nothing is written to the hard disk that indicates its presence. Code Red, Witty, and SQL Slammer are examples of worms where their presence are only evident in Memory (RAM) and not on the hard disk [1] [2] [3] [4]. The acquisition of volatile data is one of the first steps in incident handling that is executed in the containment phase. Well-known incident handling guides [5] [6] [7] [8] emphasize the importance of performing digital evidence acquisition in a timely manner based on volatility order.

Conventionally, the collection of volatile data from a system under investigation is performed through live response. This is where the first responder utilizes trusted binaries that are self-compiled with their own libraries in order to not rely completely on the compromised system in the collection of digital evidence. In fact, this method still depends on untrusted code which could have been modified by the attacker since these binaries along with their libraries use the system calls to contact the kernel. The other method of investigating volatile data is to perform a memory image analysis of the investigated system, which can be used as an alternative to live response. The later usually utilizes system administration tools to retrieve information about the system state from kernel space and it may include dumping user space tasks, whereas memory analysis captures and images all the volatile data in the memory. This method can reveal critical information that is not included in the live response approach such as hidden and terminated processes [9] [10]. Compared to the live response method, memory image analysis reduces the impact to the investigated system because it does not load additional processes on the system in the same way or to the extent that live response does. The incident handler would perform only one action, a physical memory capture, to minimize the footprint on the system in question. The compromised system is not completely depended upon to retrieve the volatile evidence, as offline memory analysis which is repeatable and allows questions to be asked later, can be carried out.

In this paper we provide a comparative analysis of the two aforementioned forensic methods that addresses the forensic capabilities and limitations of employing these methods; this analysis is restricted to the Linux environment. The criteria of the comparative analysis between live response and memory imaging frames the listed sections in this paper. The following sections are: section II, a comparison of the two methods with respect to the impact on the volatile data in memory; section III, a discussion of the experiments that investigate the volatile data which is normally gathered during live response and can alternatively be gathered from memory image; section IV, a presentation of some interesting information that is typically ignored or not easily gathered during a traditional live response but can indeed be gathered using memory image analysis; section V, an evaluation of the related work, brief due to space limitations; section VI, a conclusion with consideration of future work.

II. THE IMPACT ON INVESTIGATED SYSTEM

The kernel allocates and traces the scheduled processes in the memory until they are terminated. When the processes are terminated, their location will be unmapped to the kernel space and left as unallocated data in the memory; this can be overwritten with new loaded processes. Live response techniques cannot retrieve information about unallocated data in the memory because they rely on the kernel. Consequently, when live response tools are loaded

978-0-7695-4578-3/11 $26.00 © 2011 IEEE
DOI

and allocate space, they could overwrite unallocated data in the memory, which increases the probability of losing digital evidence. Volatile data in memory is very receptive to change, and any action on the system could change many bytes in the memory. Therefore it is acknowledged that some changes could occur during the execution of a live response toolkit and this is acceptable to some level. Considering this, the digital investigation best practice promotes prudence by emphasizing minimal footprint impact on the investigated system, as the impact could be significant. In our experiment, we investigated terminated and cached processes that could be overwritten during the live response. This will serve to highlight the impact of this method on the chain of digital evidence during an investigation.

A. Experiment Methodology

In this experiment, the percentage of the terminated and cached processes that were overwritten during the simulation of the live response method was quantified in three modes. All three modes had the same experimental environment but were associated with different memory sizes. The modes are: mode 1, mode 2, and mode 3 with corresponding memory sizes, 256 MB, 512 MB and 1024 MB (see table I). The memory size in each mode was chosen deliberately in order to test the impact of the live response toolkit in general and to test exclusively with limited memory size. However, it is worth mentioning that in a real digital investigation the forensic analyst should take into consideration the processes environment including RAM and swap space. Table II depicts the sequence of procedures performed for the experiment.

We used Linux VMware 7.1 workstation running on Ubuntu 9.1 (kernel: 2.6.31.23). The Linux host machine was equipped with an Intel (R) Core (TM) 2 CPU 6700, 2.66GHz processor and 3GB of RAM. The VMware suspend facility was utilized to create a baseline memory image for each mode before the execution of the live response toolkit and to reliably obtain a raw memory dump after the execution (VMware allows us to use .vmem as a memory image). During the simulated live response in the three aforementioned modes, all the tools used for the volatile data acquisition were binaries that are contained in most Linux distributions. These tools such as ps, top, pcat, and so on are well known among Linux administrators (see table III on the last page of this paper for complete list of the live response tools used in the experiment).

TABLE I. THE EXPERIMENT ENVIRONMENT

Mode	RAM Size	Guest Operating System	Host Operating System	Environment
Mode 1	254 MB	Ubuntu 10.4 kernel 2.6.32.28	Ubuntu 9.1 kernel 2.6.31.23	Linux VMware workstation 7.1
Mode 2	512 MB			
Mode 3	1024 MB			

TABLE II. THE EXPERIMENT PROCEDURES

(1) Allocate the memory size of guest operating system based on the mode (table I). (2) Start up the guest operating system. (3) Clear and Disable the shell command history to mitigate the false positive output from strings searching. (4) Invoke 24 processes and terminate 12 of them. (5) Suspend the guest operating system. (6) Save the memory image file (.vmem) as baseline.	**Preparation**
(7) Resume the guest operating system to the previous state. (8) Execute the live response toolkit via the shell script included in response CD.	**Execution**
(9) Suspend the target system. (10) Save the memory dump after the execution of the live response toolkit. (11) Dump the ASCII strings from both memory images, which were saved before and after the execution of live response; each dump is saved in a separate file.	**Data Collection**
(12) For both files, perform string searches for signs of the 12 terminated processes. (13) Compare the difference of the findings between the two files. (14) Go back to step one to perform the experiment in a different mode.	**Analysis**

B. Experiment Result and Discussion

This section shows the experimental result of terminated processes that were overwritten after the execution of the live response tools when compared to the baseline memory images that were taken before the execution in the three modes. Fig. 1 presents the average percentage of terminated processes that remained in memory before and after live response. In mode 3, 78 % of the terminated processes remained in the baseline image but after the execution of the live response tools, 17% of the terminated processes were impacted (overwritten). Furthermore, with less allocated memory size the likelihood of overwriting more unallocated data would increase during the live response and contribute to the impact. This is illustrated in mode 1 where the allocated memory size is small, so the system heavily reclaims the available pages to handle other loaded processes. Therefore, the forensic analyst should consider memory size as a factor when deciding to use the live response approach.

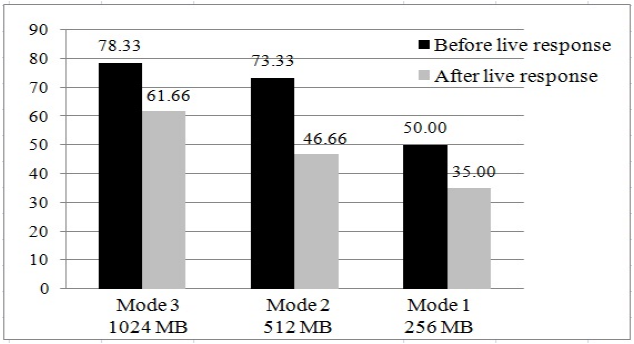

Figure 1. The average percentage of terminated processes that remained in the memory over five runs of each mode.

Further trials were performed to compare the impact of the live response verses the memory imaging method. The impact was measured in terms of the number of memory pages that changed during the simulation of each method in the three modes. In this experiment, the target system performed no user tasks, and the following steps were performed in each mode: (1) the system was booted up and a snapshot of the system state was taken as a baseline, (2) the live response tools were executed from the response CD via shell script, (3) the system was suspended to save its memory image (.vmem), (4) the system state was reverted to the baseline, (5) the dd tool was used to image the memory character device (/dev/crash), and (6) the system was suspended again to save the memory image produced by VMware (.vmem).

After collecting the memory raw images produced by VMware, we used Perl scripts to compare each byte of all the pages in the memory images from both of the methods to the baseline memory image. If there was at least one byte of change, the page was flagged as a changed page. Fig. 2 presents the average percentage of changed pages for both methods in the three modes. As a point of comparison, the memory imaging method had less of an impact on the volatile data of the investigated system. This is evident since, in all three modes, the pages which changed during the execution of this method were less than the number of pages changed during live response. Significantly, more than half of the pages in the memory were changed during live response in mode 1, while roughly a quarter of the pages were changed in the same mode during memory imaging. This demonstrates that the volatile forensic approach employed in the investigation could in some cases significantly change the system state. As digital forensics best practices emphasizes minimizing the impact to the investigated system in order to preserve the integrity of the evidence, it would be wise to dump the volatile memory first before any further action, such as employing the live response method. On the other hand, when the memory dump of the investigated system is collected, the forensic analyst can then extract the evidence directly from the memory dump instead of the live system.

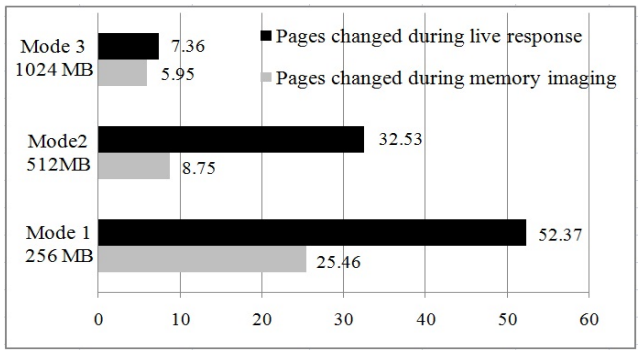

Figure 2. The average percentage of changed pages over five runs of each mode.

III. MEMORY IMAGE ANALYSIS

Memory image analysis is an area of digital forensics that has attracted many forensic researchers to develop techniques for extracting forensic artifacts from the memory dump. In the Linux environment, memory image forensics has encountered many challenges due in part to the rapid development of the Linux kernel, compilation conditions (the configurations before compiling the kernel), and the kernel customization of each distribution. All these challenges make developing a Linux memory parser tool for a specific kernel futile. However, previous research was partially successful in attempting to describe the relevant Linux memory structures that must be parsed to extract digital evidence from the memory dump with corresponding proof of concept tools [12] [13] [14].

We conducted several experiments to extract forensic artifacts from the memory dump using two memory image analysis tools:

- Volatility-Linux is an extension of Volatility framework that supports Linux memory image analysis [15]. The design of this tool makes it compilation and kernel version independent, which assists in analysing Linux memory images for a wide range of kernel versions and distributions. The tool provides plugin modules that emulate the shell commands in the live system such as ps, lsof, netstat, and so on. This helps to reconstruct the investigated system state from the memory image.
- Crash Utility is an extensible Linux core dump analysing and debugging tool. Since it is kernel-aware with source code level debugging [16], it allows the user to step through formatted kernel structures and perform low level analysis.

Each of these experiments was performed in mode 2 (refer to Table I for the experiment environment details). The VMware suspending facilities were used to obtain the raw memory dump (.vmem) and VMware Suspend State file (.vmss). The later file was used only to convert the raw memory dump to the core dump file (vmcore) in ELF format which Crash Utility understands, whereas Linux-Volatility retrieves the information directly from the memory raw image (.vmss). Both Linux-Volatility plugins and Crash Utility commands, which emulate the system utilities used in live response, were utilized during the simulation of the memory image analysis (see table III for the comparison). The memory image was taken from the system which was running different tasks beside the OS processes. From the experiments' results, the volatile data traditionally collected using live response can for the most part be collected using memory image analysis techniques.

IV. RECOVERING CACHED AND INTERNET ARTIFACTS

This section discusses experiments that have potential evidence, in particular Internet browsing artifacts and passphrases of encrypted files, which can be recovered via memory image analysis. This volatile data is ignored or not easily gathered during live response without causing significant forensic impacts on the system under

investigation. The experimental environment was the same as in section III.

A. Internet Browsing Artifacts

The internet browsing history and activities can be pivotal during digital investigation. Discovering online actions that occurred on the investigated system could provide a smoking gun that leads to more pieces of evidence. Although the browsers history and permanent cookies can be found in the hard disk, some browsing artifacts such as session cookies, logon pages, email addresses and others, are stored only in the memory. This information can be dumped from the memory in the form of ASCII strings. It is time consuming to place this information into a meaningful context, so utilizing software such as Net Analysis [17] can assist in retrieving such cached information from the memory image in an organized manner based on accessed date or URL.

We conducted an experiment to test whether the private data would be cleared from memory after closing the browser. In the target system, we used two different web browsers and adjusted their privacy settings to always clear the user's private information. We did not launch these browsers at the same time; instead we experimented on them one at a time. These browsers were browsing the same web pages and, by using Wireshark, we captured the packets between the browser and web server. The captured packets helped us to determine the parameters of the session cookies sent from the web server. We used these known variables to make the search for the session cookies in the memory easier. After the browsing sessions were completed, we closed these browsers and the memory image was taken. String searches were performed on the memory image and we were able to find these session cookies easily, as they were in html format (plaintext). Even though the browsers were closed, this information was not cleared from the memory. The browsers privacy settings are meant to clear private data such as browsing history and permanent cookies on the hard disk, but not to clear it in the memory. Furthermore, Blade Pro tool which is part of the Net Analysis package, was used to carve these web pages and their relevant images from the memory dump, and we were able to retrieve parts of these web pages.

Even the secure web pages which are encrypted on the network using SSL remain unencrypted in the memory in order to be readable for the end user. To take advantage of this, we conducted three different online transactions with different merchant websites (all connections were secured through SSL). After each completed transaction: (1) we logged out from the web account in the merchant website, (2) the browser was closed, (3) the memory was imaged, and (4) the system was rebooted. In all the three memory images, the credit card number, CVV credit card code, and the card's holder full name were found in plain text. This case also applies to received or sent emails to/from the investigated system. The forensic analyst can start looking for email addresses by using string searches for keywords such as "@<domain name>." and, if there are any, he then could search for the body of the email using some keywords such as "body=" or text between html "
" tags. In our experiment, we logged into two different web email accounts using two different email service providers. One email provider employs SSL encryption only on the login page while the other encrypts the entire connection. After these steps: (1) we sent an email from each account to different receivers, (2) logged out from both accounts, (3) closed the browser, and (4) imaged the memory; the email addresses of the senders/receivers and the body of both emails were found in plain text in the memory image. Moreover, sensitive information such as user names of both accounts and their passwords were also found in the memory image (see figure 3).

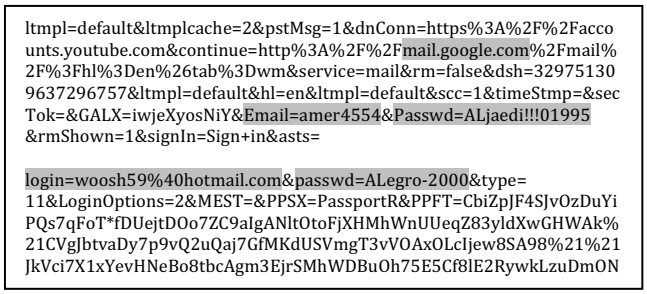

Figure 3. User names and passwords were found in memory image.

B. Passphrase of the Encrypted Files

Since attackers widely use encryption techniques to conceal their files and tools on the compromised system, this makes recovering the evidence more difficult and time consuming. While some forensic tools [18] [19] assist in recovering the encrypted evidence, memory is another resource where the investigator can search for the encryption passphrase. In a Linux environment, the passwords entered through terminal or GUI interface can be cached in the memory for many reasons, not only due to the implementation of the application itself, but also for other reasons such as the operating system caching schema, shared libraries, and buffer. Our experiment was conducted on a simulated search for passwords of encrypted files:

- The first file was encrypted through terminal compression command "zip" with password *issm_one*.
- The second file was encrypted from GUI "zip" encryption with password *issm_two*.
- The third file was encrypted though terminal GPG encryption with password *issm_three*.
- The fourth was a raw file used to mount the encrypted loop device using "losetup" terminal command, the password was *issm_four*.

After encrypting the above files, the memory was imaged. The first, third and fourth passwords were found in the memory image. This experiment shows that all the passwords typed on the terminal were found in plain text in the memory image including the root and SSH passwords. This can help the investigator to obtain the passphrase in confronting the attacker encryption methods especially when the attack involves reverse shell.

V. Related Work

Many forensic researchers and practitioners from the academic and private sectors have discussed and experimented with the forensic challenges in collecting and analysing volatile date [1] [9] [20]. The forensic challenges organized by Digital Forensic Research Workshop [21] [22] have encouraged and contributed significantly toward improving the techniques of memory image analysis. Mariusz Burdach [13] demonstrated some of the Linux memory analysis techniques and released his tool "Idetect" as a proof of concept tool. His paper provided methods for extracting information from Linux memory, through enumerating page frames of a deleted file and detecting user mode processes. However, he limited the scope of his work to 2.4.20 kernel of Linux. Jorge Mario Urrea [12] proposed additional forensic techniques for memory image analysis in Linux kernel 2.6. His thesis provided a method to integrate a swap space in volatile data forensics. Aaron Walters et al. [23] published a paper discussing the reliability and integrity of volatile data in digital forensics and the impact of live response on the investigated system. Furthermore, they introduced their tool which was initially named Volatools, and was later changed to Volatility framework [24]. Andrew Case et al. [10] [14] [15] [11] demonstrated sophisticated techniques of the memory image analysis, and presented a method of recovering the kernel structure objects of terminated tasks using the kmem_cache structure.

VI. Conclusion and Future Work

This paper compared the capabilities and limitations of two volatile data acquisition methods: live response and memory image analysis. Our experiments investigated the impact of such methods on the examined system, and showed that with live response this impact can be significant. It could overwrite potential evidence, such as terminated and cached processes, evidence that is actually ignored in live response. It was also shown that the alternative forensic method, memory image analysis, had less of an impact, which helps in mitigating the loss of evidence. Some interesting information that is not easily gathered during the traditional live response without causing much impact, and can be extracted directly from the memory dump, was presented in order to draw attention to forensic artifacts that can be recovered under memory analysis approach, but not by using live response. For future research, we would like to further investigate memory dumps to analyze not only strings but also non-string data to comprehensively quantify the impact of the forensic tools in both methods. We also plan to apply the memory image analysis techniques in detecting malware and anti-forensics methods.

Acknowledgement

The authors are thankful to the Faculty of Graduate Studies at Concordia University College of Alberta for their support in accomplishing this research. Special thanks go to Amanda Solyom and Andrew Case for their advice and discussion.

References

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

[2] M. Burdach, "Finding digital evidence in physical memory," presented at 2006 Black Hat Federal Conference. Sheraton Crystal City, Washington DC, January 2006.

[3] CAIDA Analysis of Code-Red. (2001). [Online]. Available: http:// www. caida.org/research/security/code-red/

[4] SQL Slammer worm propagation. (2003). [Online]. Available: http:// xforce .iss.net/xforce/xfdb/11153

[5] NIST Special Publication 800-61, Computer Security Incident Handling Guide, 2008. [Online]. Available:http://csrc.nist.gov/publications/nistpubs /800-61-rev1/SP800-61rev1.pdf

[6] RFC 3227 standard of evidence collection , Guidelines for Evidence Collection and Archiving. [Online]. Available: http://tools.ietf. org/html/rfc3227

[7] Incident Handling Step-by-Step and Computer Crime Investigation, In Security 504 Hacker Techniques, Exploits, and Incident Handling. SANS Institute, 2008.

[8] Electronic Crime Scene Investigation: A Guide for First Responders, 2nd ed., April 2008. [Online]. Available: http://www.ncjrs.gov/pdf files1/nij/219941.pdf

[9] C. Waits, J. Akinyele, R. Nolan, and L. Rogers, "Computer forensics: results of live response inquiry vs. memory image analysis," CERT program, CMU/SEI-2008-TN-017, August 2008

[10] A. Case, L. Marziale, C. Neckar, and G. Richard, "Treasure and tragedy in kmem_cache mining for live forensics investigation," *Digital Investigation*, vol.7, pp. S32-S40, 2010.

[11] A. Case, L. Marziale, and G. Richard, "Dynamic recreation of kernel data structures for live forensics," *Digital Investigation*, vol. 7, pp. S41-S47, 2010.

[12] J. M. Urrea, "An Analysis of Linux Ram Forensics," M.S. thesis, Dept. Computer science, Naval Postgraduate School, March 2006.

[13] M. Burdach, "Digital Forensics of the Physical Memory," unpublished, March 2005. [Online]. Available: http://strony.aster.pl/ forensics/pdf/mburdach_digital_ forensics _of_physical_memory.pdf

[14] A. Case, A. Cristina, L. Marziale, G. Richard, and V. Roussev, "FACE: Automated digital evidence discovery and correlation," Digital *Investigation*, vol. 5 , pp. S65-S75, 2008.

[15] Digital Forensics Solutions. [Online]. Available: http://dfsforensics. blogspot .com/2011/03/bringing-linux-support-to-volatility.html

[16] Red Hat Crash Utility. [Online]. Available: http://people.redhat.com/ Anderson/crash_whitepaper/

[17] Net Analysis, Digital Detective. [Online]. Available: http://www. digital-detectiv.co.uk/netanalysis.asp

[18] Access Date, Decryption and Password Cracking Software. [Online]. Available:http://accessdata.com/products/computer-forensics/decryption

[19] EnCase, New EnCase Forensic Packs Encryption Punch.[Online]. Available: http://investors.guidancesoftware.com/releasedetail .cfm?releaseid=528370

[20] D. Farmer and W. Venema, "Beyond processes" in Forensic Discovery, 1st ed. Addison-Wesley, 2004, ch. 8, pp. 161-185.

[21] DFRWS 2008 Forensics Challenge Results. [Online]. Available:http:// www.dfrws.org/2008/ challenge/ results.shtml

[22] DFRWS 2005 Forensics Challenge. [Online]. Available: http://www. dfrws .org /2005/challenge/

[23] A. Walters and N. Petroni, "Volatools: integrating volatile memory forensics into the digital investigation process," Black Hat DC 2007, February 2007.

[24] Volatile Systems website.[Online]. Available: https://www.Volatile systems.com/default/volatility#overview

. TABLE III. SUMMARY OF THE COMPARED CAPABILITIES OF LIVE RESPONSE VS. MEMORY IMAGE ANALYSIS

Forensic artifacts	Live response		Memory image analysis	
System date and time	↘	date command	↘	"xtime" symbol in system.map file has the virtual address of the time variable on a Linux system. Crash utility reports the date and the time when the memory dump was taken.
System hostname	↘	hostname command	↘	Crash utility retrieves the system name from the memory dump and **mach** Crash command reports the machine information.
IP and Mac address for the investigated host	↘	ifconfig command	↘	mpykdump extension of Crash utility (command: **xportshow -iv**) and linux_ifconfig plugin of linux- volatility.
Running processes	↘	ps -aux command	↘	linux_task_list_ps , linux_task_list_psaux plugins of Linux-Volatility and **ps**, **ps –t**, **ps –a** Crash utility commands.
Loaded kernel models	↘	lsmod command	↘	linux_lsomd plugins of Linux-Volatility and **mod** Crash utility command.
Determine scheduled task	↘	crontab and at commands		
Identify mapped drives	↘	mount command	↘	linux_mount plugin of Linux-Volatility and **mount** Crash utility command.
Examine command lines history	↘	history		
Determine what files and sockets are being accessed	↘	lsof command	↘	linux_list_open_files plugin of Linux-Volatility and **mount –f**, **foreach files** Crash utility commands.
Users logged on	↘	who, w, finger and last commands	↘	Crash utility **ps –a** command maps the user –mode tasks to their users who invoked them.
Memory general information	↘	top and vmstat commands	↘	Crash utility **kmem –i** command for general memory usage information.
System general information	↘	uname command	↘	Crash utility **mach** command for system general information.
Terminated processes			↘	String searches. Another method is inactive objects recovery [6] of caches within kmem_cache strcture by walking the free and partial lists of slab memory allocator.
Identify mounted CIFS/NFS/SAMBA shares	↘	mount and ps –aux commands	↘	**ps**, **mount** Crash commands and linux_task_list_ps, linux_mount plugins of Linux-Volatility.
Network connections and listening ports	↘	netstat and losf –i commands	↘	netstat_linux plugin of Linux-Volatility and **foreach net**, **xportshow –a** Crash utility commands.
ARP cache	↘	arp command	↘	Linux_arp plugin of Linux-Volatility and **xportshow - -arp** Crash command.
Internal routing table	↘	route command	↘	linux_route and linux_route_cache plugins of Linux-Volatility, **xportshow –r** , **xportshow –rtcache** Crash commands.
Session Cookies of browsing web pages			↘	String searches (ASCII strings in memory)
Usernames & passwords for web sessions			↘	String searches (ASCII strings in memory)
Instant messaging chat sessions			↘	String searches (ASCII strings in memory)
The recent web browsing URL,FTP			↘	Netanalysis and HstEx.
Rebuild web pages and extract individual cached files like JPEG,GIF and PNG			↘	Blade, Foremost
Dump specific process memory	↘	pcat and gcore commands	↘	gcore.so Crash extention and **vm** commands of Crash
Document data			↘	String searches (ASCII strings in memory)