The definitive guide to Linux-based live memory acquisition tools

An addendum to "State of the art concerning memory acquisition software: A detailed examination of Linux, BSD and Solaris live memory acquisition"

Richard Carbone
Certified Hacking Forensic Investigator (EC-Council)
Certified Incident Handler (SANS)
DRDC Valcartier

Sébastien Bourdon-Richard
Certified Incident Handler (SANS)
Integrated Technological Crime Unit
Royal Canadian Mounted Police

Defence R&D Canada – Valcartier

Technical Memorandum
DRDC Valcartier TM 2012-319
September 2013

Principal Author

Richard Carbone

Forensic Investigator

Approved by

Guy Turcotte

Head/Mission Critical Cyber Security Section

Approved for release by

Christian Carrier

Chief Scientist

Abstract

This technical memorandum is an addendum to TM 2012-008, “State of the art concerning memory acquisition: A detailed examination of Linux, BSD and Solaris live memory acquisition.” It examines in detail two additional software tools, Volatility’s Pmem and LiME’s Linux kernel memory drivers, both of which can be used for the memory acquisition of Linux-based live computer systems. The authors then compare them with Fmem and Second Look, the two best Linux-based memory acquisition tools as per TM 2012-008. Fmem and Second Look are analysed using the same methodology as for Pmem and LiME. This memorandum also amends information pertaining to the faulty memory acquisition of Fmem as conducted in the previous study. Additionally, certain inaccuracies were made in TM 2012-008. This specific text corrects them. As such, it should now be considered the authoritative reference concerning Linux, UNIX and BSD memory acquisition, although the experiments as conducted in TM 2012-008 will continue to remain valid. Finally, upon completing the analysis of these tools, the authors recommend the use of LiME for investigative fieldwork. However, other tool-specific recommendations are found and examined in the Conclusion.

Résumé

Ce mémorandum technique est un addenda au TM 2012-008, “State of the art concerning memory acquisition: A detailed examination of Linux, BSD and Solaris live memory acquisition”. Il examine en détail deux outils logiciels additionnels qui peuvent être utilisés pour l’acquisition de mémoire sur des ordinateurs Linux en exécution, plus spécifiquement les pilotes de mémoire du noyau Pmem (de Volatility) et LiME. Les auteurs les comparent par la suite avec Fmem et Second Look, les deux meilleurs outils d’acquisition de mémoire sous Linux selon le TM 2012-008. Fmem et Second Look ont été analysés en utilisant la même méthodologie que pour Pmem et LiME. Ce mémorandum corrige également les informations relatives à l’acquisition de mémoire fautive de Fmem telle que menée dans l’étude précédente. De plus, certaines inexactitudes ainsi portées ont été inscrites dans le TM 2012-008. Le présent texte les corrige aussi. À ce titre, il doit être considéré comme la référence faisant autorité en ce qui concerne l’acquisition de mémoire sous Linux, UNIX et BSD, bien que les expériences menées dans le TM 2012-008 demeurent valides. Finalement, après avoir complété l’analyse de ces outils, les auteurs recommandent l’utilisation de LiME pour le travail d’enquête. Cependant, des recommandations spécifiques à d’autres outils sont aussi examinées dans la conclusion.

This page intentionally left blank.

Executive summary

The definitive guide to Linux-based live memory acquisition tools: An addendum to "State of the art concerning memory acquisition software: A detailed examination of Linux, BSD and Solaris live memory acquisition"

Carbone, Richard and Bourdon-Richard, Sébastien; DRDC Valcartier TM 2012-319; Defence R&D Canada – Valcartier; September 2013.

Two additional Linux-based memory acquisition tools came out only months after the initial UNIX-based memory acquisition work was completed (TM 2012-008). The authors found it pertinent to conduct additional memory acquisition experiments against them to determine which Linux-specific memory acquisition tool (s) is (are) the most appropriate choice (s). As of this time and to the best knowledge of the authors, this document is the most detailed analysis and comparison of its type available in the public literature.

This memorandum also amends information pertaining to the faulty memory acquisition of Fmem as conducted in TM 2012-008. The original acquisition was inaccurate. Based on the new technical memory acquisition, information provided in this memorandum concerning the determination of the correct size for a Linux-based memory dump, the authors have decided that reacquiring memory using Fmem would be of benefit to all concerned parties. Moreover, certain inaccuracies were made in TM 2012-008. Therefore, in correcting them and re-conducting the Fmem-based memory acquisition experiments, the authors are confident to state that this current document can be considered the authoritative text on Linux, UNIX and BSD memory acquisition. This document supersedes the background material provided in TM 2012-008, but does not in any way negate the memory acquisition experiments conducted therein, with the exception of Fmem, which has been redone herein.

The two new Linux-specific memory acquisition tools examined are Volatility's Pmem and LiME's LKM. Both are similar in functionality to Fmem and Second Look, as seen in TM 2012-008. However, Pmem is a recent Linux LKM initiative, while LiME is geared towards the acquisition of memory running atop Linux and Linux-based devices (such as Android). The authors then compared the memory acquisition results for Pmem, LiME, Fmem and Second Look, all while considering the analysability of these memory images using the Volatility memory analysis framework. As with the original research work, the experiments were conducted against modern x86, x86 PAE and x64 Linux systems.

Based on these comparisons and analyses, the authors have determined that LiME is best suited for investigative forensic fieldwork. Moreover, they reiterate their overall findings obtained for Solaris and BSD as per TM 2012-008.

This specific work is a joint effort between Defence Research and Development Canada (DRDC) Valcartier and the Royal Canadian Mounted Police (RCMP). It was carried out over a period of several months as part of the Live Computer Forensics project, an agreement between DRDC Valcartier and the RCMP (SRE-09-015, 31XF20).

The results of this project will also be of great interest to the Canadian Forces Network Operations Centre (CFNOC), the RCMP’s Integrated Technological Crime Unit (ITCU), the Sûreté du Québec and other cyber investigation teams.

Sommaire

The definitive guide to Linux-based live memory acquisition tools: An addendum to "State of the art concerning memory acquisition software: A detailed examination of Linux, BSD and Solaris live memory acquisition"

Carbone, Richard and Bourdon-Richard, Sébastien ; DRDC Valcartier TM 2012-319 ; R & D pour la défense Canada – Valcartier; septembre 2013.

Deux autres outils d'acquisition de mémoire pour Linux ont été publiés quelques mois seulement après que le travail initial sur l'acquisition de mémoire pour Unix ait été achevé (TM 2012-008). Les auteurs ont trouvé pertinent de mener des expériences d'acquisition de mémoire supplémentaires sur ceux-ci afin de déterminer quel (s) outil (s) d'acquisition de mémoire est (sont) le plus approprié (s) pour Linux. À ce jour et au meilleur des connaissances des auteurs, ce document contient les comparaisons et analyses d'outils les plus détaillées de ce type disponibles dans la littérature publique.

Ce mémorandum corrige également les informations relatives à l'acquisition de mémoire erronée avec Fmem, telle qu'utilisée pour le TM 2012-008. Sur la base d'une nouvelle technique d'acquisition, car l'acquisition initiale était inexacte, et suite aux informations fournies dans le présent mémorandum relatives à l'évaluation de la taille exacte d'une image mémoire sous Linux, les auteurs ont décidé qu'effectuer à nouveau l'acquisition de mémoire à l'aide de Fmem serait bénéfique pour tous les partis concernés. De plus, certaines inexactitudes ont été inscrites dans le TM 2012-008. Par conséquent, en les corrigeant et en effectuant à nouveau les expériences d'acquisition de mémoire avec Fmem, les auteurs sont convaincus de pouvoir affirmer que ce document peut être considéré comme texte de référence pour l'acquisition de mémoire sur Linux, UNIX et BSD. Il remplace donc les informations de base fournies dans le TM 2012-008, mais ne nie d'aucune façon les expériences d'acquisition de mémoires dans celui-ci, à l'exception de Fmem, qui a été refaite dans le présent document.

Les deux nouveaux outils d'acquisition de mémoire spécifiques à Linux examinés sont Pmem de Volatility et LiME. Ceux-ci ont des fonctionnalités similaires à Fmem et Second Look, comme le montre le TM 2012-008. Cependant, Pmem est une initiative récente d'un module chargeable du noyau Linux, tandis que LiME est orientée vers l'acquisition de mémoire sur des appareils fonctionnant sous Linux ou basés sur Linux (p. ex. Android). Les auteurs ont ensuite comparé les résultats d'acquisition de mémoire pour Pmem, LiME, Fmem et Second Look, tout en considérant l'analysabilité de ces images mémoires en utilisant le cadriciel d'analyse mémoire Volatility. Comme pour le travail de recherche original, les expériences ont été menées sur des systèmes Linux x86, x86 PAE et x64 récents.

Sur la base de ces comparaison et analyses, les auteurs ont déterminé que LiME est le mieux adapté pour les enquêtes informatiques judiciaires. De plus, ils réitèrent l'ensemble de leurs résultats obtenus pour Solaris et BSD selon le TM 2012-008.

Ce travail spécifique est un effort conjoint entre Recherche et développement pour la défense Canada (RDDC) Valcartier et la Gendarmerie royale du Canada (GRC). Il a été réalisé sur une période de plusieurs mois dans le cadre du projet "Live Computer Forensics", qui est un accord entre RDDC Valcartier et la GRC (SRE-09-015, 31XF20).

Les résultats de ce projet seront aussi d'un grand intérêt pour le Centre d'opérations des réseaux des Forces canadiennes (CORFC), le Groupe intégré de la criminalité technologique (GICT) de la GRC, la Sûreté du Québec (SQ), ainsi que d'autres équipes d'enquête de cyberattaques.

Table of contents

List of figures

List of tables

Acknowledgements

The authors would like to thank Mr. Philippe Charland for peer reviewing this text and providing helpful comments in order to improve it.

Disclaimer and use policy

Requirements, assumptions and exclusions

It is assumed that the reader is altogether familiar with digital forensics and the various techniques and methodologies associated thereto. This technical memorandum is not an introduction to digital forensics, its techniques or methodologies. However, this technical memorandum will endeavour to present an adequate and technically oriented background to enable the reader to carry out and implement the work and analysis conducted herein.

This present work examines x86, x86 PAE and x64 based Linux, BSD and UNIX-based systems only. However, the term x86 is used interchangeably in certain portions of this text to generically represent both x86 and x86 PAE. When PAE specifics are discussed, the use of the x86 PAE is used.

This endeavour has been conducted primarily using a Windows-based system while secondary result validation was carried out using a Linux-based system. As such, regardless of the reader's own specific set-up, the reader should arrive at the same overall results as those presented herein, assuming that the guest operating systems are similarly configured and that the same virtualisation technology is used. All operating systems examined have been fully virtualised.

Two primary Windows systems were used for memory experimentation, one at DRDC Valcartier and the other at the RCMP. Technical details concerning the computer system used for experimentation at the RCMP can be found in Annex A.1, while those concerning the computer system used at DRDC Valcartier can be found in Annex A.2. Details for the technical configuration used for Fmem memory acquisition, as conducted at DRDC Valcartier, can be found in Annex A.3. The authors used their own departmentally provided computer systems to conduct the experimentation against a set of prebuilt and preconfigured VirtualBox-based virtual machines (see annexes B and C for details). As such, Mr. Bourdon-Richard carried out the LiME experiments at the RCMP, while Mr. Carbone carried out the Pmem and Fmem trials at DRDC.

All guest operating systems were tested under Oracle VirtualBox 4.2.0 (Linux and Windows version) with the appropriate VirtualBox Extension Pack installed (see Annex B and C for details). The exception to this are the Fmem memory experiments that were redone in this memorandum which continued using Oracle VirtualBox 4.1.0, as was used in the original experiments in TM 2012-008. VMware Workstation, another very popular choice for operating system virtualisation, was not used so that the reader would have not to rely on commercially licensed software in order to validate the results obtained by the authors.

It is important to emphasize that should memory acquisition of a physical, non-virtualised x86 or x64 based system fail, it may be possible to acquire that system's memory using the cold boot attack [1], since this technique is not affected by the underlying operating system. However, the success of the cold boot attack should be considered as experimental at best [2]. As such, the investigator must be realistic in his expectations for acquiring memory using this technique.

Red Hat 9 Linux is not examined in this work. In TM 2012-008, it was clearly demonstrated that it would not support and compile LKM-based modules without, at a minimum, recompiling the kernel and possibly changing portions of the source code to accommodate the various memory acquisition tools.

Target audience

This technical memorandum has been written for the computer forensic investigator who may have to perform a Linux-based memory acquisition at one time or another in the function of his or her duties. Although information exists across the Internet, bringing them together into a coherent and comprehensive manner was a matter of some undertaking on the part of the authors.

This technical memorandum is not, however, an examination of computer memory analysis. This specific field of research is outside the scope of this work and warrants an altogether separate technical discussion of the subject matter.

This page intentionally left blank.

1 Background

1.1 Context

This subsection provides the necessary context to understand how this memorandum ties in with its sister report, TM 2012-008.

1.1.1 Objective

The objective of this technical memorandum is to pick up where its sister document, TM 2012-008, left off. Specifically, the latter technical memorandum carried out an in-depth examination of Linux, BSD and Solaris-based memory acquisition, while this one is entirely Linux-specific.

However, this specific technical memorandum has a twofold purpose. Its primary objective is to examine two new Linux-specific memory acquisition tools, which only recently have gathered enough attention to make them worthwhile to examine. The second objective is to correct certain oversights conducted in TM 2012-008. Of these oversights, the Fmem memory acquisition tool will be re-examined. It has come to the authors' attention that relying on */proc/meminfo* for determining the correct upper memory address to use for memory acquisition is incorrect. Instead, */proc/iomem* should be used.

Finally, this memorandum will examine the acquisition-based experimental results obtained using Pmem, LiME and Fmem, and compare them to those obtained for Second Look (see TM 20120-008 for details) in order to determine which tool is the most suitable for field use by forensic investigators. To this end, the Volatility memory analysis framework will be used.

1.1.2 Historical note to past Linux memory forensics analyses

It is important to point out that when TM 2012-008 was written, the Volatility framework offered rudimentary Linux memory support and as such, the acquired memory images from TM 2012-008 were assessed as best as they could be. Memory analysis efforts at that time were manual in nature and were therefore more subjective. However, using string counts and automated string-based byte offset analysis, it was still possible to largely determine in an objective manner whether a given memory image was adequately populated with data and structures. Moreover, all memory images obtained thereto were visually examined for specific cues and patterns applicable only to memory dumps.

Even though Second Look offered a memory analysis framework, it was not used at that time, as it was found to be too unstable. Moreover, while the Volatility framework of that time did support Linux memory dumps, its Linux memory support was found to be unusable for most of the required analyses.

1.2 Linux operating system background

This subsection examines the background of the various Linux-based operating systems experimented upon in this work.

1.2.1 Fedora/Fedora Core Linux

Generating no revenue, Fedora Core Linux is developed in its entirety by the open source community, although it is sponsored by Red Hat Inc. Having taken up the mantle of Red Hat Linux, it continues to be freely available. Moreover, it comes available in both x86 and x64 flavours. Specifically, the x86-based version is by default an x86 PAE kernel, although the user has the ability to install an x86-only kernel instead. The x64 kernel is bundled with a full x64 distribution. Fedora Core is also a RPM-based distribution.

Despite being very popular, it continues to remain behind Ubuntu and Linux Mint in terms of its adoption [7, 8]. The distribution is considered a technology adoption leader as it continuously incorporates new capabilities as they become available in subsequent distribution releases. Its release schedule is approximately every six months.

At the time of the final revision of this memorandum, the current release was version 19, although when this work had commenced in mid-2012, version 17 was the current Fedora distribution. The very first version of Fedora was released in November 2003, just several months before Red Hat 9 became end-of-life. The history of Fedora Linux, developed by the community under the umbrella of the Fedora Project, is somewhat convoluted. It is noteworthy to state that all versions of Fedora Linux prior to version 7 are known as Fedora Core Linux, while those as of version 7 are known as Fedora Linux. Within this document and TM 2012-008, the authors refer this operating system as Fedora Core.

This work examines memory acquisition tools under both Fedora Core 15 and 17, although all results are equally applicable to Fedora 18 and 19. Fedora Core 15, although two cycles out of revision from the version of Fedora used for these experiments (version 17) was used in TM 2012-008. Fedora Core 18 and 19 were released in January and June 2013, respectively.

Red Hat continues to invest and support the Fedora Project, since it uses it as a testing ground for assessing technologies that may eventually be incorporated into Red Hat's Enterprise Linux products.

1.2.2 Ubuntu

Ubuntu is a Debian-based Linux operating system. The Ubuntu initiative is sponsored by UK technology company Canonical that is owned by South African Mark Shuttleworth. Unlike Fedora, commercially sponsored but generating no actual revenue, Ubuntu generates revenue by providing Ubuntu-related technical support and services. However, the Ubuntu operating system itself is entirely free of charge.

Perhaps due to the Ubuntu philosophy, it is currently among the most popular Linux desktops in use today [8]. Its first release was October 2004 and its release schedule is approximately every six months. The current version, 13.04, was released April 2013. However, the authors have chosen to work with Ubuntu 12.04 LTS, released in April 2012, as it will be officially supported by Canonical for a period of five years.

This work examines memory acquisition tools under both Ubuntu 11.14 and 12.04 LTS. Ubuntu 11.04, now several revisions behind the current release, was used in TM 2012-008, as neither Ubuntu 11.10 nor 12.04 were available when experimentation began under said report. However, the experiments and results as carried out in this specific memorandum are equally applicable to Ubuntu 13.04.

Ubuntu is available as both an x86 and x64 operating system and is very desktop friendly, more so than many other Linux distributions. By default, the x86-based distribution does not provide a PAE-based kernel, although compiled versions of PAE kernels are available from the Ubuntu repository for installation. Moreover, while Ubuntu is largely Debian-based, some closed source programs and drivers are bundled with it.

The Ubuntu Foundation, created in July 2005, ensures that Ubuntu will continue to remain a well-funded Linux distribution in order for the community to continue developing and supporting it. Ubuntu has also become actively involved in recent cloud computing initiatives by providing specific cloud-based technologies in its latest Ubuntu Server release.

1.3 Particulars concerning memory acquisition and analysis

This subsection examines particular issues concerning computer memory, its acquisition and volatility.

1.3.1 Forensically capturing memory

Physically capturing memory under Linux is not particularly difficult, assuming the necessary acquisition software and hardware are available. All modern Linux distributions fully support USB mass storage devices. However, correctly recognizing these devices and mounting them may at times be precarious, depending on the underlying kernel and the level of hardware support provided therein.

Of course, investigators need not be confined only to USB mass storage. Modern Linux kernels fully support FireWire standards 1394a and 1394b. However, the level of support offered by the underlying kernel is entirely dependent on its maturity. Moreover, production systems may have their support for these devices altogether disabled in order to prevent employee data exfiltration. As such, the investigator must be prepared and capable of saving memory dumps wherever necessary or possible, depending on the underlying circumstances.

If the investigator finds that USB or FireWire-based mass storage device support is either not available in the operating system or has been disabled, then memory acquisition could be carried out over the network using NFS. Since many of these systems find themselves in networked environments, NFS is often a workable solution. It is important to be aware that older versions of NFS have readily attainable upper file size limits. Newer implementations of NFS including versions 3 and 4 support 64-bit file sizes, but differing operating systems may impose other unknown constraints.

Moreover, the use of NFS requires that both the target and remote systems be configured to support it. Otherwise, the investigator will have to configure it manually. Generally, even for systems that do not have NFS enabled by default, it is not particularly difficult to get it working and should not require any system reboots. However, operating system variations may again impose differing limitations.

With the exception of LiME and Helix 3 Pro, none of the other tools examined in this work or in TM 2012-008 innately have the ability to stream memory dump-files directly across the network. Other tools can do so, but only after making use of intermediary tools (e.g. *rsh*, *ssh*, etc.). LiME is an exception in comparison to all these tools in that its network capability is directly integrated into its LKM. The investigator needs only to specify the appropriate network parameters to the LiME LKM to save a dump to another system elsewhere on the network.

The use of Windows file sharing atop Linux operating systems is not examined herein, although it is possible. The software components required for this may necessitate installation, since they are generally not considered an integral part of most Linux operating systems.

Although swap space is used by most Linux systems, it is not acquired within this work or in any of the various experiments. Unlike physical computer memory (RAM), swap space acquisition occurs like any other disk partition from a live system. As such, in so long as the investigator has root privileges, he can readily acquire all of a system's swap space.

1.3.2 Computer memory volatility

It is important to consider the volatility of computer memory when attempting to acquire it. Furthermore, it does not matter if the computer system is running DOS, UNIX, Windows, or any other operating system. The fact that an individual, in this case a computer forensic investigator, runs a memory acquisition program atop the computer system changes the state of the underlying system. This is a universal principle, commonly known as the Observer Effect, and follows through for all cases where a physical intervention is made against a given computer system [3, 4].

In the case of computer memory acquisition, in order to obtain a copy of the system's memory, the investigator must interact with the system (in order to observe it) and then run some program, command, or utility to acquire its memory. This process irreversibly changes the running state of the computer system and as such, certain bytes of information that may contain evidence may be permanently lost. However, it is logical to conclude that the more memory a given system has, the less likely this is to occur, as evidence is apt to be spread out across said memory. However, no matter the care and consistency of the steps used by the investigator, some data will inevitably be lost with no way of discerning what it was.

However, computer memory acquired through diligence should ultimately hold up to court-based challenges, in so long as the investigator understands the actions he carried out and their potential impact on the underlying system. This is, of course, where open source software shines in contrast to closed proprietary acquisition software [5].

1.4 Linux memory specifics

This subsection examines the various peculiarities concerning Linux memory management, which is unlike the memory management of Solaris and BSD.

1.4.1 Details

PAE-enabled processors allow x86 PAE-capable Linux kernels to use a 36-bit memory-addressing scheme, thereby allowing the system to address up to a maximum of 62 GiB[1] RAM [9, 10]. However, if a given distribution does not install a PAE-enabled kernel, one is usually available from the distribution's software repository, as was the case with Ubuntu, whose PAE kernel and corresponding source code headers had to be manually downloaded, installed and whose boot loader had to be reconfigured. However, these are rather trivial and well documented[2] reconfiguration operations.

Unlike the various x86 and x64 based BSD operating systems, Linux has been PAE-capable since kernel 2.3.23, released in October 1999 [9]. As such, it is common to find many x86 Linux kernels with PAE support compiled directly in, although this varies widely by distribution and user preferences. Some distributions, running 2.4.x, 2.6.x or 3.x kernels will by default, install a PAE-based kernel. Sometimes, it is set as the default bootable kernel and other times it is not. Moreover, some users prefer the use of PAE kernels while others do not. As such, it is difficult to determine whether a given x86 Linux operating system supports PAE in its current running state. However, it does make sense to use PAE-enabled kernels on any system running an x86 Linux distribution with close to or more than 4 GiB RAM, in order to make the most of available resources.

All x64 Linux kernels support more than 4 GiB RAM, unlike certain BSD distributions that by default do not. The configuration details for the Linux-based virtual machines examined herein can be found in annexes B and C. From these experiments, it is clearly demonstrated that PAE-enabled kernels do in fact detect and use memory above the 32-bit based memory limit (4 GiB RAM). As such, Linux-based PAE memory allocation is straightforward since the amount of memory supported by x86 PAE and x64 Linux systems are uniform below 62 GiB RAM.

However, direct memory acquisition under Linux is not as straightforward as it was under BSD. The reasons for this are examined next.

1.4.2 Linux memory acquisition

Memory acquisition under Linux is not particularly obvious at first glance. Unlike with BSD and Solaris operating systems, modern Linux systems no longer give direct memory access,

[1] 36-bit memory addressing can access up to a maximum of 64 GiB RAM although the last 2 GiB RAM are reserved while the first 62 GiB RAM can be used for main memory [7].

[2] The largest and most popular Linux documentation repository is The Linux Documentation Project (see http://tldp.org/ for more details).

necessitating the need for memory drivers, including but not limited to Second Look, Fmem, LiME and Pmem.

Memory device */dev/mem* is a direct interface to physical memory, while */dev/kmem* provides access to the kernel's virtual address space [17]. Although they are similar, it is advised that where both are present, memory be acquired first from */dev/mem* prior to attempting */dev/kmem*, which should only be used in the event the former fails. The ability to use these will vary by distribution and kernel version.

For a variety of reasons, direct access to physical (*/dev/mem*) and kernel memory (*/dev/kmem* and */proc/kcore*, respectively) has been limited. In 2.6.x and 3.x kernels, the restriction appears to be caused by the *CONFIG_STRICT_DEVMEM* kernel structure [11, 12, 13 and 14]. Why memory access is limited under a 2.4.x kernel (at least for Red Hat 9) is not well understood at this time, since it does not suffer from this restriction. This restriction limits the extent to which *DD* and *Memdump* can be used for memory acquisition.

Modern 2.6.x and 3.x kernels no longer have a */dev/kmem* memory pseudo-device, although */dev/mem* continues to be present. This pseudo-device was removed in 2.6.x and 3.x kernels, due to its prevalence in Linux-based rootkit attacks. Thus, by removing the device, rootkits could no longer have immediate and direct access to the kernel's memory address space. [11, 12, 13, 14 and 16]

Even though device */dev/mem* continues to be available under Linux, it is generally not possible to acquire memory beyond the first one megabyte of memory, due to the aforementioned reasons. However, under kernel 2.4.x, it is possible to acquire significantly more memory, at least under Red Hat 9, but still less than the total amount actually detected by the operating system. Availability will undoubtedly vary by kernel, its generation (2.4.x, 2.6.x or 3.x) and distribution. Linux memory acquisition-based experimental results are available in Annex D. Specifics for Red Hat 9 can be found in TM 2012-008.

Although Linux 2.4.x kernels do continue to support both */dev/kmem* and */dev/mem* memory devices, support for memory device */dev/kmem* can be reactivated for some 2.6.x and 3.x kernels by recompiling it in. Unfortunately, a concise list of which distributions permit this device's recompilation is not currently available. Notwithstanding this, for distributions shipping with kernels where this option is removed, a publicly available kernel patch reactivates this feature [15]. Experimentation conducted by the authors confirms that Fedora Linux and Ubuntu systems have their */dev/kmem* device disabled, but they can be reactivated upon selecting the appropriate kernel compilation-based configuration options.

Linux provides several kernel-specific structures for accessing system memory. These include */proc/kcore*, */proc/kallsyms* and */proc/ksyms*. Both */proc/kallsyms* and */proc/ksyms* refer to the same kernel symbol table that is used by the kernel itself and the various LKM modules. As such, this structure has limited value for memory acquisition[3]. However, */proc/ksyms* exists only under 2.4.x kernels, whereas */proc/kallsyms* has superseded the former under 2.6.x and 3.x

[3] However, this structure may have significantly more value when conducting a manual memory analysis, which is not examined herein.

kernels. Linux kernel pseudo-files */proc/ksyms* and */proc/kallsyms* provide the same functionality as the BSD and Solaris' */dev/ksyms* device. [18, 19, 20 and 21]

On the other hand, pseudo-file */proc/kcore* is a representation of physical memory stored using the ELF core file format. As such, memory dumps obtained from this pseudo-file are best left for use with the system debugger, GDB. In instances where */dev/mem* or */dev/kmem* are available, they are preferred over */proc/kcore*. The total physical length of memory from kernel structure */proc/kcore* is the size of detected memory[4] plus 4 KiB[5]. Moreover, */proc/kcore* enables the acquisition, at least to some extent, of hardware-reserved computer memory. [18, 21]

Dumping memory from */dev/mem* will result in the acquisition of hardware-reserved computer memory, while acquisition from */dev/kmem* will not.

Direct access to device and kernel structures generally requires the investigator to have root privileges on the target system.

1.4.3 Operating system-specific differences concerning */dev/mem* and */dev/kmem*

The UNIX memory device */dev/mem* is found under most major UNIX systems including all BSD, Linux and Solaris operating systems. This memory device is a direct interface to physical memory, including all hardware associated I/O[6] devices and their operating system accessible hardware memory. Although memory device */dev/kmem* is similar to */dev/mem*, it does not provide direct physical memory access. Instead, it provides an interface to the kernel's virtual address space. It is important to understand, however, that the kernel’s virtual address space memory is similar to physical memory, except that no hardware-reserved memory will be accessible through this interface.

Moreover, while */dev/mem* represents actual physical byte offsets in physical memory, */dev/kmem* does not. Instead, it is representative only of the byte offsets in the kernel's memory space.

Thus, when attempting to acquire computer memory from a UNIX-based system, it is preferred to acquire memory from */dev/mem* prior to acquire it from */dev/kmem*. Specifically, memory device */dev/kmem* should only be attempted when acquisition from */dev/mem* fails or if */dev/mem* is not available on the current system. Moreover, it is important that investigators understand that Linux 2.6.x and 3.x kernels do not by default support */dev/kmem* and */dev/mem*, typically limiting memory dumps to one MiB. As such, kernel memory interface */proc/kcore* can instead be used. However, the investigator is warned. This pseudo-device has very serious limitations.

It is important to understand that Linux pseudo-file memory interface */proc/kcore*, while similar, is not the same as memory device */dev/kmem* or */dev/mem*. Specifically, the latter provides an

[4] Detected memory size is based on the *MemTotal* value found in kernel pseudo-file */proc/meminfo*.
[5] The extra 4 KiB are for ELF data structure overhead.
[6] I/O device memory is the memory found on peripheral devices within a computer system. For example, consider video card memory, network interface buffers, etc.

interface to physical memory in ELF format, while the two former pseudo-devices provide direct access to physical memory with and without hardware-reserved memory, respectively.

1.5 Memory acquisition software background

1.5.1 LiME

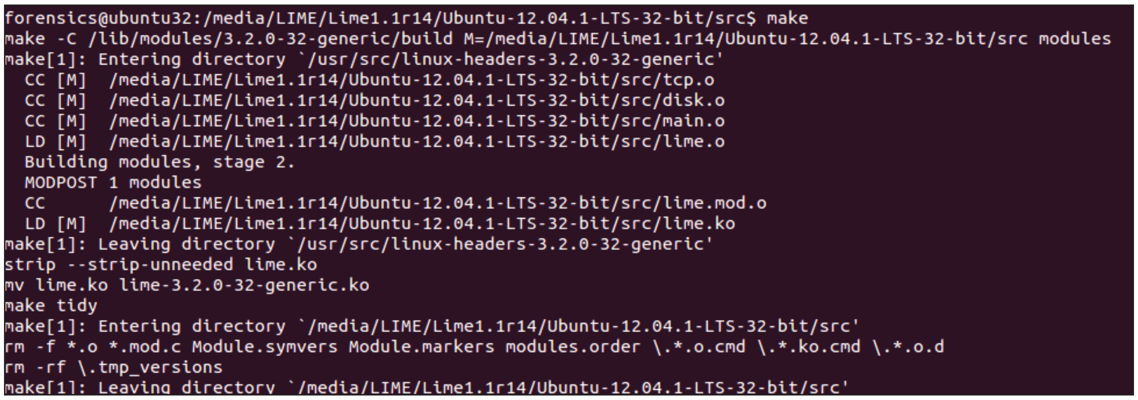
1.5.1.1 Background

LiME (Linux Memory Extractor) was developed by Joe Sylve, Andrew Case, Lodovico Marziale and Golden G. Richard III as a project to improve memory acquisition under Linux and minimize the interaction between kernel and the user space [23]. LiME is an open source LKM that allows an investigator to acquire memory against Linux and Android-based platforms. The module, operating within the kernel, enables memory acquisition to be dumped to the filesystem or over the network, without the need for intermediary network communication tools (e.g. *Netcat*). However, the scope of this evaluation does not cover memory acquisitions over the network or with Android devices. In this memorandum, version 1.1-r14of LiME was used.

Unlike other memory acquisition tools for Linux, LiME does not need a memory driver in order to map memory pages into user space or access them using tools such as *DD*. This implementation minimizes the memory footprint. Non-author specific tests comparing Fmem and LiME (formerly DMD) against a virtual Android device showed that approximately 99% of pages were correctly captured using LiME. Fmem succeeded in capturing about only 80% of said memory pages [23].

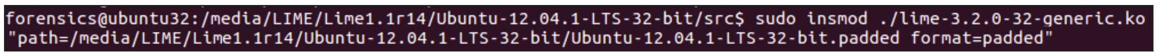
In order to capture memory, the module's source code must first be compiled (see Figure 1) and then inserted into kernel space with root privileges (see Figure 2). LiME requires two arguments in order to be correctly inserted into kernel space. These arguments are described below:

- ***path=***
  - This can be either a filename for writing the acquisition to a filesystem object or a TCP port (use format tcp:<port>) to acquire memory directly over the network.
- ***format=***
  - ***raw***
    - Acquire all "System RAM" ranges with no padding for other devices.
  - ***padded***
    - Starts from physical address 0 (zero) and pad all non-"System RAM" ranges with binary zero.
  - ***lime***
    - Acquires all "System RAM" ranges with no padding for other devices and prepend each memory address range with address space information.



```
forensics@ubuntu32:/media/LIME/Lime1.1r14/Ubuntu-12.04.1-LTS-32-bit/src$ make
make -C /lib/modules/3.2.0-32-generic/build M=/media/LIME/Lime1.1r14/Ubuntu-12.04.1-LTS-32-bit/src modules
make[1]: Entering directory `/usr/src/linux-headers-3.2.0-32-generic'
  CC [M]  /media/LIME/Lime1.1r14/Ubuntu-12.04.1-LTS-32-bit/src/tcp.o
  CC [M]  /media/LIME/Lime1.1r14/Ubuntu-12.04.1-LTS-32-bit/src/disk.o
  CC [M]  /media/LIME/Lime1.1r14/Ubuntu-12.04.1-LTS-32-bit/src/main.o
  LD [M]  /media/LIME/Lime1.1r14/Ubuntu-12.04.1-LTS-32-bit/src/lime.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /media/LIME/Lime1.1r14/Ubuntu-12.04.1-LTS-32-bit/src/lime.mod.o
  LD [M]  /media/LIME/Lime1.1r14/Ubuntu-12.04.1-LTS-32-bit/src/lime.ko
make[1]: Leaving directory `/usr/src/linux-headers-3.2.0-32-generic'
strip --strip-unneeded lime.ko
mv lime.ko lime-3.2.0-32-generic.ko
make tidy
make[1]: Entering directory `/media/LIME/Lime1.1r14/Ubuntu-12.04.1-LTS-32-bit/src'
rm -f *.o *.mod.c Module.symvers Module.markers modules.order \.*.o.cmd \.*.ko.cmd \.*.o.d
rm -rf \.tmp_versions
make[1]: Leaving directory `/media/LIME/Lime1.1r14/Ubuntu-12.04.1-LTS-32-bit/src'
```

Figure 1: LiME LKM compilation under Ubuntu 12.04 x86



```
forensics@ubuntu32:/media/LIME/Lime1.1r14/Ubuntu-12.04.1-LTS-32-bit/src$ sudo insmod ./lime-3.2.0-32-generic.ko
"path=/media/LIME/Lime1.1r14/Ubuntu-12.04.1-LTS-32-bit/Ubuntu-12.04.1-LTS-32-bit.padded format=padded"
```

Figure 2: LiME memory acquisition under Ubuntu 12.04 x86

The scope of this evaluation was limited to acquiring memory against various Linux systems directly to an external hard drive formatted using NTFS. Each Linux system was acquired using the *raw*, *padded* and *lime* formats. After compiling LiME against each test system, the commands used to acquire memory were, per system, as follows:

```
$    insmod ./lime-[VERSION].ko “path=/ext_hdd_path/mem.raw format=raw”

$    insmod ./lime-[VERSION].ko “path=/ext_hdd_path /mem.padded format=padded”

$    insmod ./lime-[VERSION].ko “path=/ext_hdd_path /mem.lime format=lime”
```

1.5.1.2 Overall impression during acquisition

LiME was stable throughout the authors-based testing and as such, they recommend its use. Investigators should use it either with the *padded* or *lime* formats. It is not recommended, however, to acquire memory using the *raw* format as memory analysis tools (e.g. Volatility) will not be able to analyze it unless the investigator pads it manually. Interestingly, the *lime* format is supported by Volatility but not by Second Look, a commercial competitor to Volatility. LiME does not require user-based tools for memory acquisition. Instead, memory is acquired at the time the LKM is loaded into kernel space.

During this evaluation and while using the *lime* format, the authors observed an odd behavior regarding the performance of the acquisition. Specifically, even if the *lime* format writes less information to the disk than the *padded* format, acquisition using this format was noticeably slower than the *padded* format. The root cause of this, however, has not been thoroughly investigated by the authors.

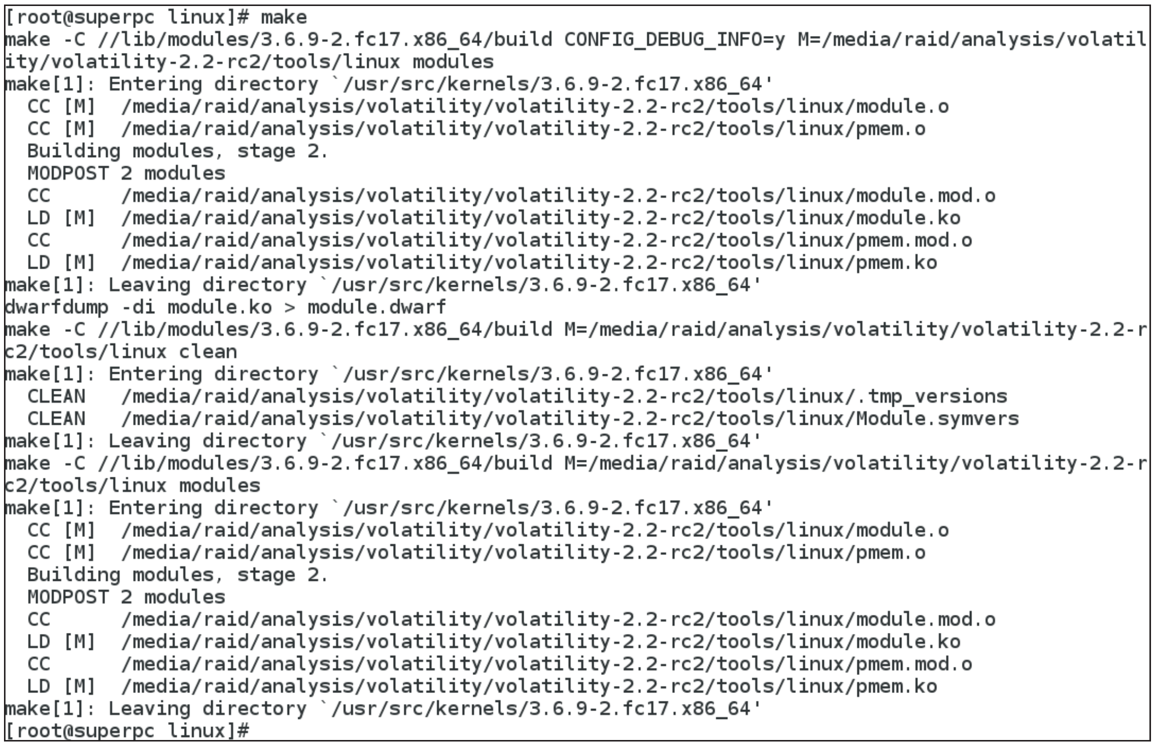
Ideally, it is recommended to use the *padded* format with LiME. Commonly used Linux memory analysis tools will support memory *padded*-based dumps, while not affecting performance (as compared to using the *lime* format).

The NTFS kernel driver was used in all LiME-based acquisitions to connect an external storage device to each virtual machine, in order to store the variously dumped memory images in a readily accessible fashion for analysis. This likely explains why all LiME acquisitions were so slow. Only this series of experiments used NTFS. See Annex D.1 for details.

1.5.2 Pmem

1.5.2.1 Background

Written by Michael Cohen of the Volatility project, the current version used in this memorandum was obtained with the source code of Volatility 2.2. Pmem is a Linux memory device driver, similar in capability to the memory drivers found in Second Look and Fmem. The driver, upon successfully loading into kernel space, provides a system-based device against which memory dumps may be obtained. A memory driver is required in order to map memory pages into user space from kernel space so that acquisition can be carried out using standard user-based tools such as *DD*.



```
[root@superpc linux]# make
make -C //lib/modules/3.6.9-2.fc17.x86_64/build CONFIG_DEBUG_INFO=y M=/media/raid/analysis/volatil
ity/volatility-2.2-rc2/tools/linux modules
make[1]: Entering directory `/usr/src/kernels/3.6.9-2.fc17.x86_64'
  CC [M]  /media/raid/analysis/volatility/volatility-2.2-rc2/tools/linux/module.o
  CC [M]  /media/raid/analysis/volatility/volatility-2.2-rc2/tools/linux/pmem.o
  Building modules, stage 2.
  MODPOST 2 modules
  CC      /media/raid/analysis/volatility/volatility-2.2-rc2/tools/linux/module.mod.o
  LD [M]  /media/raid/analysis/volatility/volatility-2.2-rc2/tools/linux/module.ko
  CC      /media/raid/analysis/volatility/volatility-2.2-rc2/tools/linux/pmem.mod.o
  LD [M]  /media/raid/analysis/volatility/volatility-2.2-rc2/tools/linux/pmem.ko
make[1]: Leaving directory `/usr/src/kernels/3.6.9-2.fc17.x86_64'
dwarfdump -di module.ko > module.dwarf
make -C //lib/modules/3.6.9-2.fc17.x86_64/build M=/media/raid/analysis/volatility/volatility-2.2-r
c2/tools/linux clean
make[1]: Entering directory `/usr/src/kernels/3.6.9-2.fc17.x86_64'
  CLEAN   /media/raid/analysis/volatility/volatility-2.2-rc2/tools/linux/.tmp_versions
  CLEAN   /media/raid/analysis/volatility/volatility-2.2-rc2/tools/linux/Module.symvers
make[1]: Leaving directory `/usr/src/kernels/3.6.9-2.fc17.x86_64'
make -C //lib/modules/3.6.9-2.fc17.x86_64/build M=/media/raid/analysis/volatility/volatility-2.2-r
c2/tools/linux modules
make[1]: Entering directory `/usr/src/kernels/3.6.9-2.fc17.x86_64'
  CC [M]  /media/raid/analysis/volatility/volatility-2.2-rc2/tools/linux/module.o
  CC [M]  /media/raid/analysis/volatility/volatility-2.2-rc2/tools/linux/pmem.o
  Building modules, stage 2.
  MODPOST 2 modules
  CC      /media/raid/analysis/volatility/volatility-2.2-rc2/tools/linux/module.mod.o
  LD [M]  /media/raid/analysis/volatility/volatility-2.2-rc2/tools/linux/module.ko
  CC      /media/raid/analysis/volatility/volatility-2.2-rc2/tools/linux/pmem.mod.o
  LD [M]  /media/raid/analysis/volatility/volatility-2.2-rc2/tools/linux/pmem.ko
make[1]: Leaving directory `/usr/src/kernels/3.6.9-2.fc17.x86_64'
[root@superpc linux]#
```

Figure 3: Pmem LKM compilation under Fedora Core 17 x64

It has come to the attention of the authors that depending on the underlying operating system, dwarf version (debugging file format) and other changes caused by various system updates, that the compilation of Pmem carried out when issuing the standard compilation command *make* may result in one or more compiler warnings. Generally, these warnings can be ignored. However, attempting to compile Pmem alone by issuing the command "*make pmem*" may prevent the compiler from presenting these warnings. This bug is related to the version of dwarf, not the compiler. Although dwarf is use by the compiler, it is not the compiler per se.

Throughout all the experiments conducted against Pmem, only when compiling it using *make* against Fedora 15 x64 with the default distribution kernel and GCC compiler did the compiler issue an error (see Annex D.2.3 for details). Specifically, the message was "ERROR: Attribute 56 (DW_AT_data_member_location)". However, specifying *make pmem* did not result in this error, as the kernel dwarf module is not compiled according to Pmem's *Makefile*. See Figure 3 for a screenshot detailing the *make*-based compilation of Pmem, where the dwarf module is also compiled. For a list of *make* options, refer to Pmem's *Makefile*.

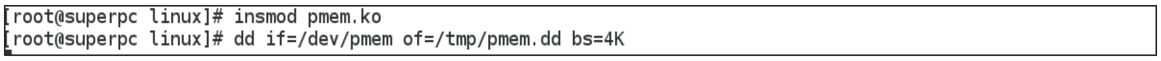
To load the driver into kernel space however, the driver's source code must be compiled and then loaded by the root user (or loaded by a user with *sudo* capability). Memory dumps can only be obtained by the root user (or user using *sudo*). Compiling the Pmem memory driver is done using the *make* command, as seen in Figure 3.

Ideally, memory dumps should be acquired in the following fashion using 4 KiB sized pages, as seen below:

```
$ dd if=/dev/pmem of=/capture_device/dump.pmem bs=4K count=MEM_IN_KIB/4KIB
```

MEM_IN_KIB is the last addressable "System RAM" address obtained from */proc/iomem*. The abovementioned command is sufficient for directly reading from memory and dumping it to a user-designated file. Since the correct block size and count parameters have been specified, the final size of the memory dump should be equal to the product of the specified byte size and count parameters.

However, this ideal memory dumping method was not used by the authors in order to test the tool's robustness. Instead, they used the last command shown in Figure 4:



```
[root@superpc linux]# insmod pmem.ko
[root@superpc linux]# dd if=/dev/pmem of=/tmp/pmem.dd bs=4K
```

Figure 4: Pmem memory acquisition under Fedora Core 17 x64

Using the aforementioned command, the authors were able to determine if the memory driver could stop dumping memory upon reaching the detected operating system memory limit. As it turns out, this was in fact the case. Only PAE Linux-based systems failed to acquire correctly the underlying system's memory.

However, Pmem did not work "out of the box." Instead, a minor source code change was required. Specifically, the following change was made to file *./volatility-2.2/tools/linux/module.c*: as shown next:

```
Line 70        #include <linux/net_namespace.h>
```

This line was changed to:

```
Line 70        #include <net/net_namespace.h>
```

Implementing this single change enabled the Pmem kernel driver to compile correctly across Ubuntu 11.04 and 12.04 (x86, x86 PAE and x64, respectively) as well as Fedora Core 15 and 17 (x86 PAE and x64, respectively).

1.5.2.2 Overall impression during acquisition

Pmem is easier to use than LiME, as it has only one dump format. However, unlike LiME, which worked against all the target operating systems, Pmem did not. Specifically, it did not work against x86 PAE-based systems. As such, is not suitable for use in the field. For detailed acquisition information, consult Annex D.2.

1.5.3 Fmem

1.5.3.1 Background

The technical details of Fmem memory acquisition have already been fully examined in TM 2012-008. Additional details concerning the correct use of (*/proc/iomem* over */proc/meminfo*) for determining the appropriate upper memory address to use for acquisition is examined in Annex E.2.

Because the original experiments in TM 2012-008 were carried out against Ubuntu 11.04 (x86, x86 PAE and x64) and Fedora 15 (x86 PAE and x64), all Fmem memory acquisition experiments conducted in this work were carried out against these very same operating systems only. Acquisition and experimentation against Ubuntu 12.04 and Fedora 17 have not been done in order to maintain consistency between the aforementioned report and this one.

1.5.3.2 Overall impression during acquisition

Based on the experimental results obtained in Annex D.3, Fmem was found to be fast and accurate. It was able to acquire memory up to the maximum limit of the physical RAM as defined by */proc/iomem*. This was done without error or issue. Thus, it can be considered suitable for field use.

The authors have conducted the experiments using Fmem 1.6-1, the same version used in TM 2012-008. However, the tool has not changed since that time.

1.5.4 Second Look

For details concerning the background and use of Second Look, consult TM 2012-008 and Annex E.1, respectively. Volatility-based memory analysis conducted herein has been conducted against the original Second Look memory images acquired as per TM 2012-008.

By default, Second Look will attempt to use the system's kernel crash driver, if it is present. If not, then the investigator must compile Second Look's provided crash driver, *pmad.c*, in order to carry out a memory dump.

1.5.4.1 Overall impression during acquisition

Memory acquisition using Second Look proved to be fast and easy. By far, Second Look is the fastest memory acquisition tool examined both in TM 2012-008 and herein. Moreover, it is able to acquire the correct amount of memory as per */proc/iomem*. Acquisitions for all operating systems succeeded without issue.

1.6 A note about the Linux kernel crash driver

The Linux kernel crash driver or LKM, found precompiled on supporting systems as *crash.ko*, is loaded into kernel space using the *modprobe* command if the investigator does not know where it is located or using the *insmod* command otherwise. It is another memory access driver (or LKM) for directly accessing the computer system's physical memory. It does not appear to be included in the standard Linux kernels. Instead, it seems to be available only with Red Hat-based Linux distributions including, but not necessarily limited to, Red Hat and Fedora (the authors have not confirmed its existence under CentOS). However, other distributions (such as Ubuntu) may also have this feature supported by applying external patches to the distribution's kernel source code [11].

Upon having been loaded into kernel space, the crash LKM creates */dev/crash*, a temporary new device that can be used to read from memory or dump it. The device can be removed from kernel space by unloading it using the *rmmod* command.

Some confusion exists between the Linux crash driver and the Linux crash project, both of which are distinct yet interdependent, although both were written by David Anderson of Red Hat. The Linux crash driver, while not officially a part of the Linux kernel, is found distributed with modern Red Hat-based distributions. Moreover, it is compiled with the stock kernels provided through these distributions and it can be used in tandem with the Linux crash project. Recall that modern 2.6.x and 3.x kernels no longer support */dev/kmem* (by default) and limited access is provided to */dev/mem*. Thus, using */dev/crash*, the Linux crash framework has complete and unmitigated access to the system's physical memory.

In contrast to the Linux crash driver, the crash project is an endeavour to provide a system crash-based analysis framework. This framework has the ability to debug a live running system or investigate the cause of system kernel panic that was saved to a core-file or dump-file for post-mortem analysis. The framework has the ability to read memory from */dev/mem*, */proc/kcore* and */dev/crash*, but only from the latter if the crash LKM has been loaded into kernel space. Recall that the ability to read from these memory devices will vary, as examined in Section 1.4.3. Many distributions, including Ubuntu, have repository crash packages available for installation, but they do not include the kernel crash driver. However, it can be applied as a kernel patch.

The authors have prepared a table that examines all the operating systems experimented upon in both this work and in TM 2012-008 to validate if the Linux kernel crash driver, *crash.ko*, is available to the investigator for immediate use (e.g. memory capture). Virtual machines and guest operating system specifics can be found in Annex B.

Table 1: Operating system support for the Linux kernel crash driver

Operating system	Compiled kernel crash driver
Ubuntu 11.04 x86	Not included / not available
Ubuntu 11.04 x86 PAE	Not included / not available
Ubuntu 11.04 x64	Not included / not available
Ubuntu 12.04 x86	Not included / not available
Ubuntu 12.04 x86 PAE	Not included / not available
Ubuntu 12.04 x64	Not included / not available
Red Hat 9.0	Not included / not available
Fedora 15 x86 PAE	Compiled and included in kernel source code
Fedora 15 x64	Compiled and included in kernel source code
Fedora 17 x86 PAE	Compiled and included in kernel source code
Fedora 17 x64	Compiled and included in kernel source code

As can be seen from this table, only the Red Hat and Fedora-based distributions actually support the */dev/crash* pseudo-device without the need for patching and recompiling the underlying kernel. Thus, based on this information, the authors recommend that investigators do not use this mechanism for memory capture, as it may not always be present.

However, in order to read and dump memory from */dev/crash*, the correct memory address offsets and ranges must be used, as per */proc/iomem*. Otherwise, attempting to address memory that is non-addressable as “System RAM” will result in a read error. This is due to this memory being non-RAM memory. Specifically, it is hardware memory and cannot be directly accessed by the operating system or kernel.

1.7 Memory impact of kernel crash drivers

The LKM or drivers found with Fmem, Pmem and Second Look's Pmad are all various forms of Red Hat's Linux kernel crash driver, albeit each with its own distinct peculiarities. It is known that the Linux kernel crash driver is well suited for kernel debugging in the event of system or kernel panic, where the kernel's memory space is written out to a corefile. However, it is not currently known if these crash drivers are forensically sound in the way they acquire memory from a given system. No research work has been published to date concerning this subject matter and as such, the authors cannot conjecture further with regard to their forensic reliability.

The extent to which use of a LKM or kernel memory driver impacts memory have not been studied, neither by the authors nor in the available literature. Thus, the authors cannot state with certainty what the potential impact may be of using these drivers and the user space tools required to dump memory, e.g. *DD*. However, the implementation of LiME was designed to minimize its memory footprint because it does not need a memory driver in order to map memory pages into user space and dump it using tools such as *DD*.

2 Memory acquisition issues and concerns

2.1 The Volatility framework

This subsection examines the various issues surrounding the use of the Volatility memory framework.

2.1.1 Using Volatility to determine the integrity of the acquired memory dumps

In previous memory acquisition work (TM 2012-008 and TM 2011-215), the main author used string counts[7], highest byte offsets[8] and visual inspection of memory dumps as his primary methods of determining if a given memory dump appeared to be valid. This was because memory analysis tools such as Volatility provided little functional support for Linux. Only since Volatility 2.2 has Linux support become more robust. Specifically, this particular version of Volatility is capable of performing various analyses against Linux-based memory dumps. Besides Volatility, another option examined in TM 2012-008 was Second Look. While it worked well against some memory images, it failed with others.

Using Volatility 2.2 and through the generation of Linux-based kernel profiles, as examined in [22], it is possible to objectively determine whether a given memory dump image is intact and faithfully represents the contents of memory. In this work, a Linux-based memory image is deemed successfully analysed using Volatility if a process listing succeeds against it using Volatility's *linux_pslist* plugin.

For example, to examine an Ubuntu 11.04 x86-based memory dump using the Volatility *linux_pslist* plugin, the following command can be used:

```
$ python vol.py -f Ubuntu-11.04-32-bit.lime --profile=LinuxUbuntu-11_04-32bitsx86
linux_pslist
```

This command instructs the Volatility framework to perform a Linux-based process listing using a specific Linux kernel profile. An example of this can be seen in Figure 5.

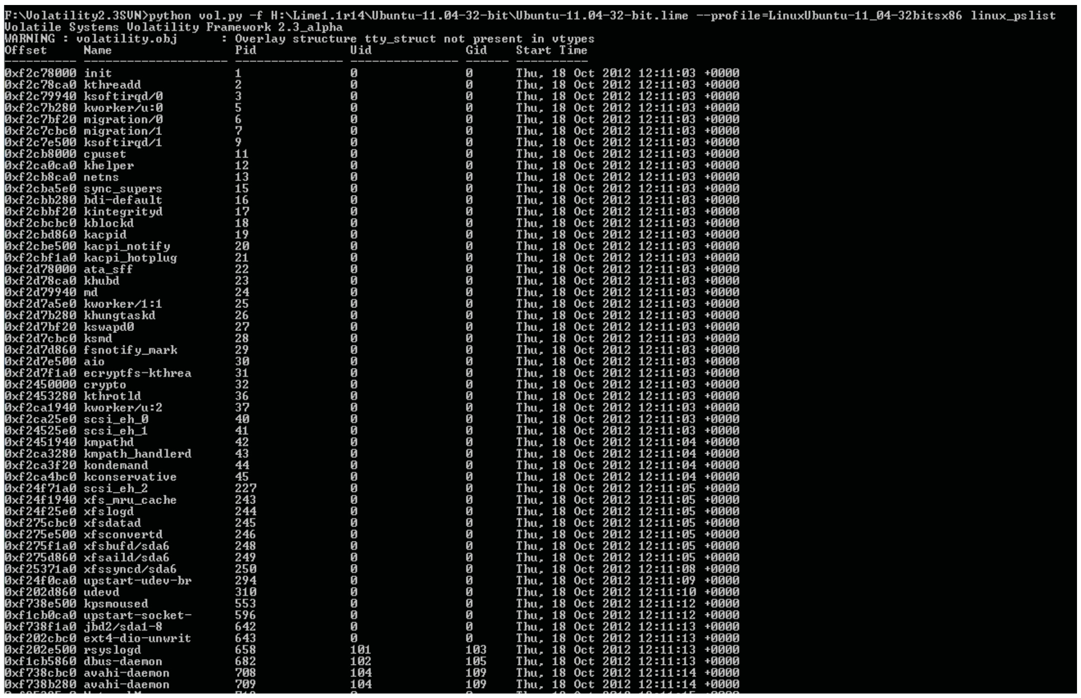
Unlike for Windows-based memory analyses using Volatility, Linux system profiles must be generated prior to analysis. Volatility includes Windows profiles[9] dating back to Windows 2000 and as recent as Windows 7 and Server 2008, both for 32 and 64-bit (Windows 8 and Windows Server 2012 memory support is not yet available in Volatility 2.2). However, complicating

[7] String counts included 7, 8, 16 and 32-bit strings as determined using the UNIX *strings* command.

[8] Highest byte offset is based against the highest string found a given bit depth. For example, the last 7-bit string found within a given memory dump image would be considered the highest byte offset for that specific bit depth.

[9] Currently, Volatility 2.2 does not support Windows 8 systems. However, newer non-production versions of Volatility may include support for Windows 8 based systems.

matters for the generation of Linux distribution specific profiles is the fact that most distributions offer regular kernel updates, as compared to Windows that does not. Thus, even if the Volatility developers were to include default Linux profiles, they would be of limited benefit to investigators, as Linux systems vary widely. Fortunately, Linux profile generation is not particularly difficult and the Volatility developers have provided information for doing so [24].



```
F:\Volatility2.3SVN>python vol.py -f H:\Lime1.1r14\Ubuntu-11.04-32-bit\Ubuntu-11.04-32-bit.lime --profile=LinuxUbuntu-11_04-32bitsx86 linux_pslist
Volatile Systems Volatility Framework 2.3_alpha
WARNING : volatility.obj      : Overlay structure tty_struct not present in vtypes
Offset     Name                 Pid             Uid             Gid    Start Time
---------- -------------------- --------------- --------------- ------ ----------
0xf2c78000 init                 1               0               0      Thu, 18 Oct 2012 12:11:03 +0000
0xf2c78ca0 kthreadd             2               0               0      Thu, 18 Oct 2012 12:11:03 +0000
0xf2c79940 ksoftirqd/0          3               0               0      Thu, 18 Oct 2012 12:11:03 +0000
0xf2c7b280 kworker/u:0          5               0               0      Thu, 18 Oct 2012 12:11:03 +0000
0xf2c7bf20 migration/0          6               0               0      Thu, 18 Oct 2012 12:11:03 +0000
0xf2c7cbc0 migration/1          7               0               0      Thu, 18 Oct 2012 12:11:03 +0000
0xf2c7e500 ksoftirqd/1          9               0               0      Thu, 18 Oct 2012 12:11:03 +0000
0xf2cb8000 cpuset               11              0               0      Thu, 18 Oct 2012 12:11:03 +0000
0xf2ca0ca0 khelper              12              0               0      Thu, 18 Oct 2012 12:11:03 +0000
0xf2cb8ca0 netns                13              0               0      Thu, 18 Oct 2012 12:11:03 +0000
0xf2cba5e0 sync_supers          15              0               0      Thu, 18 Oct 2012 12:11:03 +0000
0xf2cbb280 bdi-default          16              0               0      Thu, 18 Oct 2012 12:11:03 +0000
0xf2cbbf20 kintegrityd          17              0               0      Thu, 18 Oct 2012 12:11:03 +0000
0xf2cbcbc0 kblockd              18              0               0      Thu, 18 Oct 2012 12:11:03 +0000
0xf2cbd860 kacpid               19              0               0      Thu, 18 Oct 2012 12:11:03 +0000
0xf2cbe500 kacpi_notify         20              0               0      Thu, 18 Oct 2012 12:11:03 +0000
0xf2cbf1a0 kacpi_hotplug        21              0               0      Thu, 18 Oct 2012 12:11:03 +0000
0xf2d78000 ata_sff              22              0               0      Thu, 18 Oct 2012 12:11:03 +0000
0xf2d78ca0 khubd                23              0               0      Thu, 18 Oct 2012 12:11:03 +0000
0xf2d79940 md                   24              0               0      Thu, 18 Oct 2012 12:11:03 +0000
0xf2d7a5e0 kworker/1:1          25              0               0      Thu, 18 Oct 2012 12:11:03 +0000
0xf2d7b280 khungtaskd           26              0               0      Thu, 18 Oct 2012 12:11:03 +0000
0xf2d7bf20 kswapd0              27              0               0      Thu, 18 Oct 2012 12:11:03 +0000
0xf2d7cbc0 ksmd                 28              0               0      Thu, 18 Oct 2012 12:11:03 +0000
0xf2d7d860 fsnotify_mark        29              0               0      Thu, 18 Oct 2012 12:11:03 +0000
0xf2d7e500 aio                  30              0               0      Thu, 18 Oct 2012 12:11:03 +0000
0xf2d7f1a0 ecryptfs-kthrea      31              0               0      Thu, 18 Oct 2012 12:11:03 +0000
0xf2450000 crypto               32              0               0      Thu, 18 Oct 2012 12:11:03 +0000
0xf2453280 kthrotld             36              0               0      Thu, 18 Oct 2012 12:11:03 +0000
0xf2ca1940 kworker/u:2          37              0               0      Thu, 18 Oct 2012 12:11:03 +0000
0xf2ca25e0 scsi_eh_0            40              0               0      Thu, 18 Oct 2012 12:11:03 +0000
0xf24525e0 scsi_eh_1            41              0               0      Thu, 18 Oct 2012 12:11:03 +0000
0xf2451940 kmpathd              42              0               0      Thu, 18 Oct 2012 12:11:04 +0000
0xf2ca3280 kmpath_handlerd      43              0               0      Thu, 18 Oct 2012 12:11:04 +0000
0xf2ca3f20 kondemand            44              0               0      Thu, 18 Oct 2012 12:11:04 +0000
0xf2ca4bc0 kconservative        45              0               0      Thu, 18 Oct 2012 12:11:04 +0000
0xf24f71a0 scsi_eh_2            227             0               0      Thu, 18 Oct 2012 12:11:05 +0000
0xf24f1940 xfs_mru_cache        243             0               0      Thu, 18 Oct 2012 12:11:05 +0000
0xf24f25e0 xfslogd              244             0               0      Thu, 18 Oct 2012 12:11:05 +0000
0xf275cbc0 xfsdatad             245             0               0      Thu, 18 Oct 2012 12:11:05 +0000
0xf275e500 xfsconvertd          246             0               0      Thu, 18 Oct 2012 12:11:05 +0000
0xf275f1a0 xfsbufd/sda6         248             0               0      Thu, 18 Oct 2012 12:11:05 +0000
0xf275d860 xfsaild/sda6         249             0               0      Thu, 18 Oct 2012 12:11:05 +0000
0xf25371a0 xfssyncd/sda6        250             0               0      Thu, 18 Oct 2012 12:11:08 +0000
0xf24f0ca0 upstart-udev-br      294             0               0      Thu, 18 Oct 2012 12:11:09 +0000
0xf202d860 udevd                310             0               0      Thu, 18 Oct 2012 12:11:10 +0000
0xf738e500 kpsmoused            553             0               0      Thu, 18 Oct 2012 12:11:12 +0000
0xf1cb0ca0 upstart-socket-      596             0               0      Thu, 18 Oct 2012 12:11:12 +0000
0xf738f1a0 jbd2/sda1-8          642             0               0      Thu, 18 Oct 2012 12:11:13 +0000
0xf202cbc0 ext4-dio-unwrit      643             0               0      Thu, 18 Oct 2012 12:11:13 +0000
0xf202e500 rsyslogd             658             101             103    Thu, 18 Oct 2012 12:11:13 +0000
0xf1cb5860 dbus-daemon          682             102             105    Thu, 18 Oct 2012 12:11:13 +0000
0xf738cbc0 avahi-daemon         708             104             109    Thu, 18 Oct 2012 12:11:14 +0000
0xf738b280 avahi-daemon         709             104             109    Thu, 18 Oct 2012 12:11:14 +0000
```

Figure 5: Example process listing from a Linux memory dump using Volatility

2.1.2 Issues using Volatility

The authors determined early on that the Volatility 2.2 framework had specific support related issues with respect to Fedora-based operating systems, which can be found annexes D and G. The exception to this is Volatility 2.3 SVN revision r2574, which works against Fedora *lime* and *padded*-based memory dumps. However, all other Fedora-based memory dumps obtained using Pmem, Fmem and Second Look were found to be non-functional when analysed with Volatility.

In cases where Volatility did not succeed in examining a given memory image, string counts of 7, 8, 16 and 32 bits were conducted against said memory images in order to verify whether it was sufficiently populated with data and structures. Moreover, string byte offsets analysis of the aforementioned memory images lent additional credence to a memory image's intactness.

The LiME-based acquisitions were analysed with Volatility 2.3 SVN r2754. Interestingly, even though Fedora memory dumps could not typically be analysed using Volatility 2.2 or 2.3, the *lime* and *padded* memory dumps obtained under Fedora 15 and 17 (x86, x86 PAE and x64) were found to be analysable under Volatility 2.3 SVN r2574. However, it must be stated that Volatility 2.3 SVN r2754 lacked adequate support for Fedora 15 x64-based memory dumps. The secondary author proposed a modified version for Volatility's file *dwarf.py* to rectify this specific issue[10]. This fix was found to work and was used herein for all Volatility 2.3 SVN r2574-based analyses.

A full listing of all memory dumps obtained herein and the specific versions of Volatility (2.2 or 2.3 SVN r2574) used against them can be found in Annex G. This annex provides a detailed comparison of which dumps could be successfully analysed using Volatility. This information may help serve the reader in his own attempts with Linux-based memory acquisition analysis.

2.1.3 About generating kernel profiles

Although Volatility does not include many Linux kernel profiles, in order to overcome this, the Volatility team has provided documentation for generating Linux kernel profiles. This information can be found at http://code.google.com/p/volatility/wiki/LinuxMemoryForensics.

However, using Linux kernel profiles for Fedora-based systems was at times problematic even though these Fedora profiles were generated correctly as per the Volatility team's instructions. In certain circumstances, when using a Fedora-based kernel profile, Volatility would not function correctly, as documented in Annex D as per the analyses found in Section 3. For more information, the secondary author's Volatility message board posting can be found at https://code.google.com/p/volatility/issues/detail?id=355#c4.

2.2 Examining memory dump sizes and other issues

This subsection examines how memory dumps, as based on their size, should be considered as complete or inadequate. This is not a straightforward task, as it depends largely on the underlying operating system and architecture.

Although this specific memorandum directly examines Linux memory acquisitions (LiME, Pmem and Fmem), it is nevertheless a continuation of TM 2012-008, and as such, will address not only determining the appropriate memory dump size for Linux, but also for x86 and x64 based Solaris and BSD operating systems.

2.2.1 The PCI hole problem and memory acquisition

The examination of the "PCI Hole" issue [25] was not directly examined in TM 2012-008. This problem, commonly known as the 3 GiB barrier, has been plaguing x86-based operating systems for years. The only way to avoid it is to migrate to an x64 architecture and operating system.

In essence, the problem is related to the manner in which PC-based computer memory (RAM[11]) found between 3 and 4 GiB is mapped out for usage exclusively by the system's hardware.

[10] For more information, consult the secondary author's post available at https://code.google.com/p/volatility/issues/detail?id=367.

[11] RAM denotes the physical memory modules inserted into a computer's motherboard and does not take into account processor cache, video card memory or any other form of memory-based computing technology.

Computer memory is mapped linearly. Fortunately, almost all RAM and some I/O[12] memory is accessible or addressable[13], in one form or another, from the operating system kernel. However, the kernel does not typically make all the RAM available to non-kernel processes, and even then, it is not known[14] if all kernel processes will have access to it. Thus, under modern Linux systems, it is only through a kernel memory driver that this kernel memory can be fully accessed and acquired.

Ultimately, the problem relates to how RAM and I/O memory are mapped out in a typical x86-based computer system. RAM is allocated in linearly mapped blocks, found interspersed among I/O memory (TM 2012-008 provides a depiction of this in its Annex D against an Intel i7 980X based system equipped with 24 GiB RAM). Testing this and many other physical and virtual machines using three specific operating systems, namely Linux x86, x86 PAE and x64, resulted in differing results with respect to the mapping of physical RAM ("System RAM") and of the underlying I/O hardware. Specifically, under x86 Linux, RAM was available up to 3,583.94 MiB while under x86 PAE-based Linux, it is available up the 62 GiB memory limit. Under x64 Linux, no practical limitations are in effect.

More specifically, a 32-bit operating system will be able to address RAM up to that system's 32-bit memory limit which is always less than 4 GiB RAM. The distinction must be made between the operating system, user-land tools and applications, and the kernel, which always has full access to all computer-based RAM. However, in order to access RAM above the 32-bit operating system limit, a memory device driver is required, which in some cases is provided through system pseudo-devices. These specifics are examined within this discussion.

Thus, the PCI hole problem affects different operating systems in various ways, all of which depends on the underlying kernel, how it maps hardware into memory and its ability to provide direct memory access. Thus, it can be concluded that based on experimentation, PC operating system support for x86, x86 PAE and x64 memory addressing differs according to the underlying kernel in use. These results apply equally to Linux, Solaris, BSD and even Windows NT-based systems (XP, Vista, Windows 7, etc.).

It is important to be aware that some operating systems, specifically BSD variants, provide direct physical memory access via kernel memory device */dev/mem*. However, this pseudo-device is only accessible to the root user. BSD's */dev/mem* device behaves precisely as a memory driver. In tests carried out in TM 2012-008, all the x86 BSD variants were able to fully acquire their host virtual machine's memory including the hardware-reserved portion of RAM, as per a manual analysis of each system's memory dump. Unfortunately, no x86 PAE-based BSD systems were

[12] I/O memory denotes memory belonging to peripheral devices such as disk caches, buffers on network interfaces, video card memory, etc.

[13] There are limitations to this, however. For example, a video card with 1 GiB onboard card graphics generally makes only a portion of its overall memory available to the kernel. Nevertheless, portions of it are available to the system.

[14] More research is required in order to determine to what extent various kernel processes of Linux, Solaris and BSD have full and unfettered access to this memory. Documentation is too sparse and inconsistent to draw and firm conclusions at this point. Only full source code analysis of the various kernels and (continued from Footnote 15) subsystems will answer this question, and this requires an in-depth knowledge of kernel design and implementation.

available for testing, as PAE-enabled kernels must be compiled after the installation of a given BSD distribution and as such, this approach was not undertaken by the authors. It was also determined that x64 BSD systems could also have their full physical memory acquired, with the exception of x64 OpenBSD, which for the tested version therein, did not support 64-bit memory allocation.

The reader must understand that without being fully versed in the finer details of operating system kernel and virtual memory management internals, including the underlying platform's hardware architecture, it is impossible to determine definitively how memory will be mapped and managed. Frankly, few people know these details intricately enough and fewer still have written about it outside of hardware manufacturers' engineering manuals. Moreover, quality publicly available literature is even rarer and both hardware and operating system vendors do not typically provide this level of detail in their system engineering guides obtained when purchasing their systems. Thus, the authors are basing their assertions in this section on both the very limited available literature, the various memory drivers' source code (Fmem, Second Look, Pmem and LiME) and other memory acquisition programs (Memdump and Helix 3 Pro R3).

As with BSD, UNIX systems such as Solaris were found to provide a readily accessible memory device, */dev/mem*, which provides unmitigated memory access for x64 Solaris systems, but not for x86 PAE[15] systems, as based on *Memdump*-based memory acquisitions. Specifically, when running an x64 Solaris kernel, no difficulty was encountered during memory acquisition. Thus, direct memory access under Solaris appears superior to that of Linux, at least when comparing x64-based Solaris and Linux systems.

However, for x86-based Solaris PAE systems, as based on tests conducted in TM 2012-008, it was determined that acquisition against a system ceased when the memory dump grew to 3,583 MiB in size. This appears to indicate that memory beyond this limit was dedicated to the system's hardware and was made inaccessible by the kernel, as it was entirely available under the x64 kernel. Thus, at least for x86 PAE Solaris, memory between 3,583 MiB and 4 GiB will likely remain inaccessible. It is known that both the x64 and x86 PAE systems recognized exactly the same amount of memory, as based on results obtained from Solaris' *prtconf* command and from David W. Noon's C code program [26]. The former command recognized 8,192 MiB while the latter program recognized 8,191.559 MiB RAM. Tests against *prtconf* appear to indicate that this program provides the full amount of physical RAM while Noon's code provides the amount of memory seen by the kernel.

Thus, taking into account the fact that under BSD, *Memdump* could fully acquire memory from the underlying x86-based system, including hardware-reserved memory between 3 and 4 GiB, it can be reasonably concluded that Solaris has a built-in mechanism which prevents access to this region of memory in its x86 PAE incarnation. However, since there were no x86 PAE instances of BSD to compare against, the authors do not want to draw too many conclusions. Specifically, since *Memdump* was compiled as an x86 program under x86 PAE Solaris, it could not, by its very design, access memory beyond 4 GiB without changing its 32-bit *read ()* calls to *pread64 ()* or *llseek ()* and *read ()*. The reader may ask why not use *DD*? This was tried in TM 2012-008 and the results were utterly disappointing.

[15] Modern x86-based Solaris no longer provides standard 32-bit memory addressing (non-PAE) kernels.

2.2.2 Defining the correct memory dump size for BSD

Based on the experimentation conducted in TM 2012-008, the following recommendations are in line with this report's results. BSD memory acquisition, at least for x86 and x64 operating systems, should be considered complete when the memory dump is the same size as the amount of physical RAM, as reported by the virtual machine's settings or the computer's BIOS. It is possible that the memory dump will continue to grow beyond this size. If this occurs, then extract only the amount of memory that corresponds to the physical RAM of the underlying system from that memory image.

Moreover, experimentation confirms that memory acquisition of x86 BSD systems will acquire the hardware-reserved portion of physical RAM. Regarding x86 PAE kernels for BSD systems, no recommendations are available at this time.

Finally, since direct memory access is available via */dev/mem*, *DD* and *Memdump* can be used, but preference should be given to *Memdump*.

2.2.3 Defining the correct memory dump size for Solaris

Under x86 and x64 Solaris, the Solaris kernel is able to see and access all the system's physical RAM, except a very small portion several hundred KiB in size which is reserved[16]. Using *Memdump*, the only successful memory acquisition tool tested for Solaris, the investigator can expect x64 Solaris memory acquisitions to be the same size as that reported by Noon's C program [26]. Where x86 PAE acquisitions are concerned, the memory dumps to be acquired are expected to be up to the hardware-reserved memory limit of x86, which can vary according to the underlying hardware. However, an accurate estimate would be that the latter half of the fourth GiB of RAM would be set aside for hardware. Acquisition of memory beyond 4 GiB for x86 PAE systems is possible. However, it requires that the *Memdump* tool is modified to use an x64 *read ()* or some other system call that has the ability to go beyond the limitations of the standard 32-bit read *()* used by *Memdump*.

Obviously, these results do not apply to non-x86 based Solaris. 32-bit Solaris on SPARC will not follow the same memory addressing ranges due to hardware-based architectural differences. Moreover, Solaris, like BSD, does not provide a Linux-like */proc* subsystem with which to query the system about its memory ranges. Instead, various operating system-specific diagnostic suites such as SunVTS can be used.

2.2.4 Defining the correct memory dump size for Linux

Under Linux, the memory reported by */proc/iomem* is always larger than the memory reported by */proc/meminfo*. This is because */proc/iomem* reports memory based upon the actual addressing of physical memory (RAM) and other I/O memory, whereas memory reported by */proc/meminfo* is

[16] For a detailed discussion of this reason, please refer to the examination of ACPI and BIOS INT 0x15 EAX=0xE820 as examined in more detail in Section 2.2.4, specifically the discussion concerning reserved mapped memory for the ACPI and BIOS.

based upon the memory the kernel reserves for the system. The memory reported by */proc/meminfo* never equals the full amount of physical RAM allocated to the system. However, when adding up the "System RAM" memory address ranges as per */proc/iomem* (see calculations and example below for more information), the memory reported therein is equal or very close to the actual amount of physical memory.

There is always a difference between the amounts of memory reported by */proc/iomem* and */proc/meminfo*, with the latter being the smaller of the two. This difference is due to the kernel reserving memory for itself that is not made available to the rest of the operating system. The exact contents of this kernel-reserved memory are not fully understood at this time, but it appears to contain only kernel subsystems. Moreover, this kernel-reserved memory does not contain hardware-reserved memory, as based on many observations conducted by the authors using x86, x86 PAE and x64 virtual machines with varying size of memory. In the authors' observations, a difference in memory sizes was maintained between */proc/iomem* and */proc/meminfo*.

In order to better understand this memory difference, consider an Ubuntu 12.04 x86 with exactly 4 GiB of allocated RAM running within a virtual machine. It reports 3,616,096 KiB RAM from *MemTotal* as per */proc/meminfo* while the last addressable RAM memory address according to */proc/iomem* is 3,669,952 KiB. This 53,856 KiB memory difference has been reserved by the kernel for its own subsystems. Moreover, based on the addressable memory range as seen in */proc/iomem*, memory between 3,669,952 KiB and 4,194,304 KiB is not available, even though the x86 virtual machine has been allocated exactly 4 GiB RAM. This is altogether normal, as it was set aside for hardware-reserved memory (memory reserved for the computer's hardware) and under an x86 Linux operating system, this region is off-limits.

Thus, when conducting a memory acquisition against this x86 Linux operating system using a memory driver or LKM, all physical memory up to 3,669,952 KiB should be acquired. Of course, x86 Linux systems equipped with less than 4 GiB RAM will have differing amounts of available memory and the location and address ranges of memory set aside for hardware-reserved memory will vary accordingly.

Acquisition against x86-based Linux PAE and x64 systems should be acquired up to the upper memory "System RAM" address as per */proc/iomem*. However, since the other non-Linux UNIX-based systems examined in TM 2012-008 do not provide a Linux-like */proc* subsystem, complete memory acquisitions for them are different, as examined in sections 2.2.2 and 2.2.3.

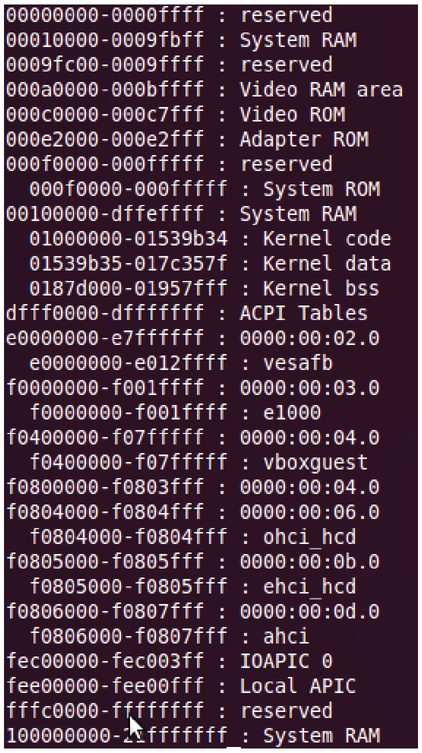
The highest accessible "System RAM" page reported by */proc/iomem* should be used to define the correct dump size for Linux systems. The logic behind this assertion is that on Linux, only */proc/iomem* provides the true memory ranges for RAM. When adding up these ranges manually (see the calculation below), they add up to the full amount of physical RAM minus several hundred kibibytes reserved for the BIOS and ACPI mappings (this is examined further on in this text). Thus, a memory dump that respects */proc/iomem* will dump all RAM and pad with zeroes all non-RAM I/O memory ranges, as done by all the tools examined in this work (Pmem, LiME *padded* dumps, Second Look and Fmem). However, at an absolute minimum, a memory dump that is equal to the total size of the RAM memory (e.g. LiME *raw* dump) ranges as per */proc/iomem* is also acceptable, although it is not the preferred dump size as it does not see memory in quite the same way the CPU does.

In order to analyze memory dumps and correctly convert virtual to physical linear-based memory addressing, memory analysis software must have the same view as the CPU does as per [27]. This is the true reason why a memory dump should be equal to the size of the highest accessible "System RAM" page reported by */proc/iomem*. An example of this can be found in the calculation below and in Figure 6. This calculation is based on an x64 virtual machine allocated with exactly 8 GiB RAM. It would have the following "System RAM" */proc/iomem* output:

```
9FBFFh + 1 - 10000h =                 588,800
DFFEFFFFh + 1 - 100000 =        3,756,982,272
21FFFFFFFh + 1 - 100000000 =    4,831,838,208
---------------------------------------------
Total "System RAM" =            8,589,409,280
```

Thus, a system allocated with exactly 8 GiB RAM (equal to 8,589,934,592 bytes) will not have exactly that same amount of memory seen available as "System RAM." This is shown in the above calculation as the addition of the "System RAM" memory ranges actually adds up to 8,589,409,280 bytes of physical RAM. There is a very small difference between the two (exactly 513 KiB or 525,312 bytes). This difference is related to the memory map defined by the BIOS (BIOS INT 0x15 EAX=0xE820[17]). The memory map defines what the different regions of physical memory are used for. Some regions are reserved and cannot be used by the operating system. For example, the BIOS can reserve memory for Real Mode IVT (Interrupt Vector Table), BDA (BIOS data area) and EBDA (Extended BIOS Data Area) [28].

[17] See http://www.brokenthorn.com/Resources/OSDev17.html for more information.



```
00000000-0000ffff : reserved
00010000-0009fbff : System RAM
0009fc00-0009ffff : reserved
000a0000-000bffff : Video RAM area
000c0000-000c7fff : Video ROM
000e2000-000e2fff : Adapter ROM
000f0000-000fffff : reserved
  000f0000-000fffff : System ROM
00100000-dffeffff : System RAM
  01000000-01539b34 : Kernel code
  01539b35-017c357f : Kernel data
  0187d000-01957fff : Kernel bss
dfff0000-dfffffff : ACPI Tables
e0000000-e7ffffff : 0000:00:02.0
  e0000000-e012ffff : vesafb
f0000000-f001ffff : 0000:00:03.0
  f0000000-f001ffff : e1000
f0400000-f07fffff : 0000:00:04.0
  f0400000-f07fffff : vboxguest
f0800000-f0803fff : 0000:00:04.0
f0804000-f0804fff : 0000:00:06.0
  f0804000-f0804fff : ohci_hcd
f0805000-f0805fff : 0000:00:0b.0
  f0805000-f0805fff : ehci_hcd
f0806000-f0807fff : 0000:00:0d.0
  f0806000-f0807fff : ahci
fec00000-fec003ff : IOAPIC 0
fee00000-fee00fff : Local APIC
fffc0000-ffffffff : reserved
100000000-21fffffff : System RAM
```

Figure 6: Example output from /proc/iomem from a system with 8 GiB RAM

In the previous example, 459,776 bytes (1,048,576 – 588,800) were reserved in the "Low" memory region (e.g. < 1 MiB). In addition, 65,536 bytes were reserved for the ACPI table (*/proc/iomem* memory addresses found between 0xDFFF0000 – 0xDFFFFFFF) as seen in the figure below. There is a total of 525,312 bytes (459,776 + 65,536) of reserved RAM and these memory ranges cannot be used by the OS nor displayed as "System RAM" by */proc/iomem*. Thus, in a relatively straightforward manner, using *DD* and a Linux memory driver or LKM, it is possible to acquire precisely[18] all physical RAM memory ranges as per */proc/iomem*.

Incidentally, under x64 Solaris, this is the same reason why a very small portion was not accessible to *Memdump*. The difference between the amount of physical memory and the size of the memory dump-file was 462,848 bytes (as per TM 2012-008) and this occurred for the same reasons enumerated above, with the exception that the Solaris kernel was able to "see" slightly more of these mappings than the Linux kernel.

In order to validate which Linux memory acquisition-based software tool is capable of acquiring memory up to the limit defined by */proc/iomem*, a comparative analysis has been conducted herein. This study compares the best Linux acquisition tools as determined by TM 2012-008 (Second Look) against those examined herein (LiME, Pmem and Fmem).

[18] This includes the *bs*, *skip* and *count* parameters (lesser used parameters can also be used).

2.2.5 Memory mapping

Unlike other operating systems that limit the investigator's ability to query the system concerning its memory and hardware I/O mapping, Linux makes this information readily available through */proc/iomem*. Other systems such as BSD and Solaris do not directly provide this ability. On these systems, in order to query memory and hardware I/O mapping, third party or custom kernel querying software is required. Under Windows, the registry can be queried for this information, but it is not obvious to locate.

To provide a more thorough understanding of memory mapping, several examples will be used. Consider the example of an Ubuntu 11.04 x86-based operating system allocated exactly 4 GiB RAM whose memory is mapped at the following locations:

> 00010000 – 0009fbff or 8FBFF bytes = 588,799 bytes
>
> 00100000 – dffeffff or DFEEFFFF bytes = 3,756,982,271 bytes
>
> Total memory = 3,757,571,070 bytes or exactly 3,583 MiB

Now the example of an Ubuntu 11.04 x86 PAE-based operating system allocated exactly 4 GiB RAM whose memory is mapped at the following locations:

> 00010000 – 0009fbff or 8FBFF bytes = 588,799 bytes
>
> 00100000 – dffeffff or DFEEFFFF bytes = 3,756,982,271 bytes
>
> 100000000 – 11fffffff or 1FFFFFFF bytes = 536,870,911 bytes
>
> Total memory = 4,294,441,981 bytes which is 525,315 bytes shy of exactly 4 GiB

Finally, consider the example of an Ubuntu 11.04 x64-based operating system allocated exactly 4 GiB RAM whose memory is mapped at the following locations:

> 00010000 – 0009fbff or 8FBFF bytes = 588,799 bytes
>
> 00100000 – dffeffff or DFEEFFFF bytes = 3,756,982,271 bytes
>
> 100000000 – 11fffffff or 1FFFFFFF bytes = 536,870,911 bytes
>
> Total memory = 4,294,441,981 bytes which is 525,315 bytes shy of exactly 4 GiB

Thus, based on these examples, it is easy to see and understand why x86 PAE and x64 RAM-based memory addressing is the same, up to the PAE limit of 62 GiB. However, beyond the PAE limit of 62 GiB RAM, an x64 operating system must be used.

While understanding the amount of RAM available to a given operating system, be it x86, x86 PAE or x64, it is important to differentiate between memory allocation and memory mapping. Memory allocation refers to the amount of memory a given computer system, physical or virtual in nature, has access to while memory mapping refers to the linear mapping of address ranges representing RAM and hardware I/O. Thus, even though in some memory mappings RAM goes beyond the size of its allotment, this by no means indicates that the system has been somehow overcommitted.

More specifically, looking back at the previous example of Ubuntu 11.04 x64, the final memory address for its physical memory is 0x11fffffff (or 4,831,838,207 bytes). This virtual computer system certainly does not have this amount of memory or "System RAM" available as exactly 4 GiB RAM (or 4,294,967,296 bytes) was allocated to it. What the reader must consider is that all hardware in a computer system has its own memory. Some of this memory is derived from specific hardware I/O components mapped into the operating system by the kernel, while some of it is altogether unavailable. In addition, even if hardware memory is addressable, it does not guarantee its availability to the kernel or its subsystems.

Thus, understanding how these address ranges tie into the expected memory dump sizes a given memory acquisition is quintessential to quantifying that dump's suitability for use in an investigation.

Unfortunately, when investigating BSD and Solaris-based systems, because they both lack similar mechanisms to */proc/iomem*, it is not possible to determine how their memory is specifically mapped.

However, based on [27], it becomes apparent that it is in fact the underlying BIOS and processor that maps the various memory regions, including hardware I/O memory and RAM, and that this process has nothing to do with the operating system kernel. Instead, the kernel must manage memory through its virtual memory manager and ensure that memory (i.e. RAM and hardware I/O memory), is made available to the appropriate subsystem or user application. Thus, the kernel maps the various physical memories into the operating system for use by the various subsystems. Consequently, all PC operating systems have their memory mapped the same way. However, the manner in which the kernel portrays this memory to the underlying operating system and subsystems is an altogether different subject far outside the scope of this work.

Finally, these memory mappings will vary from system to system according to the underlying hardware, chipsets, processors, associated I/O components and other attached peripherals.

3 Experimental outcomes and evaluations

3.1 Results and analyses

This section examines in detail the experimental results found in Annex D and the analyses conducted in annexes F and G. Where possible, attempts were made to provide meaningful insight into the various results obtained herein. Moreover, the following analyses provide brief commentaries and recommendations for each tool.

3.1.1 LiME

This subsection examines LiME-specific memory acquisition and the subsequent analysis of acquired memory dumps using Volatility 2.3 SVN r2754. Additional information concerning LiME can be found in Section 1.5.1 and annexes D.1, F.1 and G.1.1.

3.1.1.1 Technical background

In order to determine if a given LiME memory dump successfully acquired all of a system's memory, the dump size was compared to the system's memory map as established by the kernel's memory range as per */proc/iomem*. For example, consider the memory range for the Ubuntu 11.04 x86-based test system:

```
$    cat /proc/iomem | grep "System RAM"
     00010000 - 0009FBFF : System RAM
     00100000 - DFFEFFFF : System RAM
```

These ranges translate to:

65,536 to 654,335

1,048,576 to 3,758,030,847

These ranges are informative for LiME in order to instruct it as to which address it should stop dumping memory from. Of course, this depends on the type of LiME memory dump desired by the investigator, as examined in the following three subsections.

3.1.1.1.1 Raw memory dump

During the memory acquisition experiments, the authors considered a successful *raw* memory dump to be equivalent to the sum of the size of the "System RAM" ranges established by */proc/iomem*. For example, considering the aforementioned Ubuntu 11.04 x86-based test system, to determine if a *raw* memory dump was successful, the sum of the "System RAM" ranges must be added together to establish a final memory size. Thus, this Ubuntu system yields the following accessible memory ranges as shown next:

0009FBFF + 1 – 00010000 = 8FC00 (588,800 bytes)

DFFEFFFF + 1 – 00100000 = DFEF0000 (3,756,982,272 bytes)

Together, these two ranges yield a total of 3,757,571,072 bytes of acquirable system memory.

3.1.1.1.2 Padded memory dump

Unlike a *raw*-based memory acquisition, a *padded* memory dump is complete when the dump is equivalent to the size of the last memory address range, as determined by examining */proc/iomem* ("System RAM" based entries). For example, when considering the aforementioned Ubuntu 11.04 x86-based system, a *padded* memory dump is successful when it is equal to the size of the last abovementioned memory system-based range, as seen below:

DFFEFFFF + 1 = DFFF0000 (3,758,030,848 bytes)

Thus, for the Ubuntu 11.04 x86-based system, a successful *padded*-based memory acquisition would be 3,758,030,848 bytes in size. This type of dump is called padded because the memory ranges that, according to */proc/iomem*, contain non-System RAM memory are padded with binary zero.

3.1.1.1.3 Lime memory dump

Finally, a successful *lime*-based memory acquisition occurs when a given memory dump is equivalent to the sum of all the acquirable system memory ranges, as per "System RAM" found by examining */proc/iomem*, plus a fixed-sized header of 32 bytes for each individual memory range. For example, consider the Ubuntu 11.04 x86-based system. A complete memory dump, as per the sum of the aforementioned memory ranges was found to the 3,757,571,072 bytes in size. However, since two memory ranges are involved, two 32-byte headers are to be added to the dump's size as shown below:

0009FBFF + 1 – 00010000 = 8FC00 (588,800 bytes)

DFFEFFFF + 1 – 00100000 = DFEF0000 (3,756,982,272 bytes)

2 x 32 byte headers = 64 bytes

This yields a final memory dump size of 3,757,571,136 bytes.

3.1.1.2 Memory acquisition results

The various LiME modules compiled and loaded without incident for each operating system examined herein. However, acquiring and analysing LiME memory acquisitions is more complicated than for any other memory acquisition tool examined throughout this work. This is because LiME has three memory acquisition modes, as examined in the preceding subsection. When examining the table below and the experimental results found in Annex D.1, the technical background concerning LiME memory dumps as found in Section 3.1.1.1 must be considered to establish whether a given memory dump was successful. It was found the LiME memory acquisition succeeded with one caveat, i.e. it was consistently one memory page (4 KiB) short in acquiring a given PAE-based system's memory.

For all other memory dumps, the acquired memory dumps were of the expected size. Specifics for the *raw*, *padded* and *lime* memory dumps can be found in the table below, as based on the results obtained in Annex D.1.

Table 2: LiME-based experimental results summary for memory acquisition-based dump size

Operating System	***Raw* format**	***Padded* format**	***Lime* format**
Ubuntu 11.04 x86	Complete	Complete	Complete
Ubuntu 11.04 x86 PAE	Missing 4,096 bytes	Missing 4,096 bytes	Missing 4,096 bytes
Ubuntu 11.04 x64	Complete	Complete	Complete
Ubuntu 12.04 LTS x86	Complete	Complete	Complete
Ubuntu 12.04 LTS x86 PAE	Missing 4,096 bytes	Missing 4,096 bytes	Missing 4,096 bytes
Ubuntu 12.04 LTS x64	Complete	Complete	Complete
Fedora Core 15 x86 PAE	Missing 4,096 bytes	Missing 4,096 bytes	Missing 4,096 bytes
Fedora Core 15 x64	Complete	Complete	Complete
Fedora Core 17 x86 PAE	Missing 4,096 bytes	Missing 4,096 bytes	Missing 4,096 bytes
Fedora Core 17 x64	Complete	Complete	Complete

3.1.1.3 Analysis using Volatility

Analysis of the LiME *padded* and *lime* dumps succeeded in all cases for all the various operating systems against which memory was acquired, even though the PAE-based memory dumps were one memory page short (4 KiB). LiME memory dumps were examined using Volatility 2.3 SVN r2754. Only LiME's *raw* memory dumps were not examined herein as it is not currently supported by Volatility.

Volatility therefore succeeded in providing a complete process listing for each memory dump image, including all Ubuntu and Fedora based memory images.

It is important to note that in order for Fedora 15 x64 to be supported by Volatility, the secondary author had to modify *dwarf.py*, as previously mentioned in Section 2.1.2.

For more information concerning Volatility-based analysis, refer to Annex G.1.1 and Section 2.1.3.

3.1.1.4 Recommendations

The authors highly recommend the use of LiME. Investigators should consider using the *padded* format as it can be analysed using known Linux memory analysis tools (Volatility and Second Look) and is as fast as or faster than the *lime* format to acquire. The fact that LiME is consistently one memory page short for PAE-based operating systems does not appear to pose any problems to memory analyses of these memory dump images.

3.1.2 Pmem

This subsection examines Pmem-specific memory acquisition and the subsequent analysis of its memory dumps using Volatility 2.2 and 2.3 SVN r2574. Additional details concerning Pmem can found in Section 1.5.2 and annexes D.2, F.1 and G.1.2.

3.1.2.1 Memory acquisition results

The Pmem source code had to be modified for each operating system in order for it to compile correctly, as examined in Section 1.5.2.1. Although the source code was successfully compiled for every operating system examined using Pmem, an error was raised by the compiler for Fedora 15 x64 (see Annex D.2.3 for more information). Nevertheless, the module did produce a functional LKM for said operating system.

In acquiring memory from each operating system, it was discovered that Pmem had significant acquisition-based problems when dumping memory from PAE-based systems. This issue was without regard to the specific distribution as it occurred for both Ubuntu and Fedora-based operating systems.

Not once did Pmem result in a memory dump with the expected size. Instead, issues were found for each memory dump. Even though memory acquisition appeared to succeed at first glance against Ubuntu 11.04 x86, it was quickly discovered that this memory dump was over 58,000,000 bytes short of a complete memory dump. Recall that a complete memory dump is based upon */proc/iomem*. Furthermore, it was soon discovered that all memory dumps against the x86 and x64 systems were consistently one byte short, indicating a persistent acquisition bug innate to Pmem.

Memory acquisition specifics for Pmem can be found in Annex D.2, but they have been summarised in the following table.

Table 3: Pmem-based experimental results summary for memory acquisition

Operating System	Memory dump size	Comments
Ubuntu 11.04 x86	Missing 58,044,416 bytes	This dump was short by 58,044,416 bytes. This error is likely due to a bug with Pmem.
Ubuntu 11.04 x86 PAE	Missing 8,589,934,593 bytes	This dump failed. Far too little memory was acquired to be of use with Volatility.
Ubuntu 11.04 x64	Missing 1 byte	This dump was short by 1 byte. This error is likely due to a bug with Pmem.
Ubuntu 12.04 LTS x86	Missing 1 byte	This dump was short by 1 byte. This error is liked due to a bug with Pmem.
Ubuntu 12.04 LTS x86 PAE	Missing 8,589,934,593 bytes	This dump failed. Far too little memory was acquired to be of use with Volatility.
Ubuntu 12.04 LTS x64	Missing 1 byte	This dump was short by 1 byte. This error is likely due to a bug with Pmem.
Fedora Core 15 x86 PAE	Missing 8,589,934,593 bytes	This dump failed. Far too little memory was acquired to be of use with Volatility.
Fedora Core 15 x64	Missing 1 byte	This dump was short by 1 byte. This error is likely due to a bug with Pmem.
Fedora Core 17 x86 PAE	Missing 8,589,934,593 bytes	This dump failed. Far too little memory was acquired to be of use with Volatility.
Fedora Core 17 x64	Missing 1 byte	This dump was short by 1 byte. This error is likely due to a bug with Pmem.

3.1.2.2 Analysis using Volatility

Analysis of Pmem-based memory dumps was conducted using both Volatility 2.2 and 2.3 SVN r2574. They were both found to be effective against Ubuntu x86 and x64-based memory dumps only. Efforts to analyse memory images originating from Fedora-based systems failed as these memory dump images were incomplete. Moreover, since all memory images obtained using Pmem against x86 PAE-based Ubuntu and Fedora systems were altogether incomplete, no overall assessment concerning Volatility's analysis capabilities against x86 PAE-based Fedora and Ubuntu systems could be determined as this time.

Analysis of Fedora x64 memory images (versions 15 and 17, respectively) failed against both Volatility 2.2 and 2.3 SVN r2574. However, it must be noted that analysis using both Volatility frameworks with Fedora 17 x64 never actually failed. Instead, it failed to output any analyses after more than 10 hours of processing, which essentially is considered a failure. Thus, there was no manner in which to get either framework to function correctly with Fedora without rewriting the supporting Volatility code. Upon having completed the various experiments and analyses, the secondary author discovered that the reason Volatility could not analyse the intact[19] Fedora Pmem-based memory images was due to the improper generation of kernel-based profiles as examined in Section 2.1.3.

Using string and byte-offset analyses for the x86 PAE memory images were altogether inconclusive. However, analyses for the other two Fedora x64 systems (versions 15 and 17, respectively) indicated that these memory images appeared intact and populated with data and structures.

For more information concerning Volatility-based analysis, refer to Annex G.1.2.

3.1.2.3 Recommendations

Pmem, although a somewhat capable memory acquisition tool, is not yet ready for field use. More specifically, it should not be used for forensic acquisition against any Linux system that is running a PAE-enabled kernel. Doing so will likely result in an incomplete memory dump image.

However, it could be used against x86 and x64 Linux running 2.6.x and 3.x generation kernels. Nevertheless, the authors are of the opinion that better memory acquisition tools exist.

3.1.3 Fmem

This subsection re-examines the Fmem memory acquisitions (as carried out anew in this work) and their subsequent analysis using Volatility 2.2 and 2.3 SVN r2754. Additional information concerning Fmem is found in annexes D.3, E.2, F.1 and G.1.3.

3.1.3.1 Memory acquisition results

The Fmem module compiled without error or warning for all operating systems it was tested against. Memory acquisition was straightforward. The investigator had only to execute the script *run.sh* in order to load correctly the compiled module into kernel space at which time the memory dump could be initiated as per Annex E.2. Unlike Pmem and LiME, Fmem produced memory dump images that were all the expected size.

Memory acquisition specifics for Fmem can be found in Annex D.3, but they have been summarised in the following table.

[19] Fedora 15 and 17 x86 PAE Pmem-based memory dumps resulted in incomplete acquisitions that were entirely insufficient for analysis with Volatility or any other memory analysis framework. Only Fedora based x64 Pmem acquisitions were intact.

Table 4: Fmem-based experimental results summary for memory acquisition

Operating System	**Memory dump size**	**Comments**
Ubuntu 11.04 x86	Complete	Memory dump is the correct size and completed without issue
Ubuntu 11.04 x86 PAE	Complete	Memory dump is the correct size and completed without issue
Ubuntu 11.04 x64	Complete	Memory dump is the correct size and completed without issue
Fedora Core 15 x86 PAE	Complete	Memory dump is the correct size and completed without issue
Fedora Core 15 x64	Complete	Memory dump is the correct size and completed without issue

3.1.3.2 Analysis using Volatility

Fmem-based Volatility memory analysis, using version 2.2, succeeded without issue for the various memory dumps obtained against Ubuntu 11.04 x86, x86 PAE and x64.

Analysis of Fedora 15 x86 PAE and x64 memory images failed against both Volatility 2.2 and 2.3 SVN r2574. It is likely that the only way to get the kernel profile to work would have been to rewrite the supporting Volatility code. Based on a discovery by the secondary author examined in Section 2.1.3, it was realized that Volatility could not analyse the Fedora-based Fmem memory images. This was due to the improper creation of the kernel profiles, as based on the Volatility team's current instructions pertaining to kernel profile generation. However, an examination of the Fedora 15 memory images using strings and byte-offsets indicated that they appeared intact and populated with data and structures.

For more information concerning Volatility-based analysis, refer to Annex G.1.3.

3.1.4 Second Look

This subsection examines Second Look-specific memory acquisitions (as conducted in TM 2012-008) and the subsequent analysis of said memory dumps using Volatility 2.2 and 2.3 SVN r2754. Details can be found in Section 1.5.4 and annexes D.4, F.1 and G.1.4.

3.1.4.1 Memory acquisition results

Although the memory acquisition experiments for Second Look were conducted in TM 2012-008, the analysis of these acquisitions as per the aforementioned report was not particularly detailed. Thus, based on the results obtained in Annex C.5 as per TM 2012-008, it can be said that the *pmad* LKM for Second Look compiled without issue and was readily loaded into kernel space. Memory acquisition using the provided *secondlook-memdump.sh* acquisition script succeeded in dumping the expected amount of memory as per */proc/iomem*. In memory acquisition experiments using Second Look, it was found that its acquisitions were by far the fastest of the tools compared herein.

Memory acquisition specifics for Second Look can be found in Annex C.5 of TM 2012-008, but they have been summarised in the following table.

Table 5: Second Look-based experimental results summary for memory acquisition

Operating System	**Memory dump size**	**Comments**
Ubuntu 11.04 x86	Complete	Memory dump is the correct size and completed without issue
Ubuntu 11.04 x86 PAE	Complete	Memory dump is the correct size and completed without issue
Ubuntu 11.04 x64	Complete	Memory dump is the correct size and completed without issue
Fedora Core 15 x86 PAE	Complete	Memory dump is the correct size and completed without issue
Fedora Core 15 x64	Complete	Memory dump is the correct size and completed without issue

3.1.4.2 Analysis using Volatility

Second Look Volatility 2.2 memory analysis succeeded without issue for the various memory dumps obtained against Ubuntu 11.04 x86, x86 PAE and x64.

Analysis of Fedora 15 x86 PAE and x64 memory images failed against both Volatility 2.2 and 2.3 SVN r2574. There was no manner in which to get the Fedora kernel profiles working without rewriting Volatility's code base. After having completed the various Fedora memory acquisitions and analyses, the secondary author discovered that the reason Volatility could not support these memory images was due to the improper creation of the kernel profiles as examined in Section 2.1.3. However, examination of the Fedora 15 memory images using strings and byte-offsets indicated that they appeared intact and populated with data and structures.

For more information concerning Volatility-based analysis, refer to Annex G.1.4.

4 Conclusion and final tool assessment for Linux, BSD and Solaris UNIX

4.1 Solaris

Where Solaris x64 memory acquisitions are concerned, Memdump is the tool of choice. It works as expected for x64-based Solaris systems. However, when x86-based systems are encountered, memory acquisition will transpire up to the memory address where hardware-reserved computer memory is located, often found between 3.3 to 3.5 GiB RAM. Experimentation thus far has clearly demonstrated that memory acquired using this tool is valid and intact. However, it is not known if this tool will provide the same results atop SPARC-based systems although it is very likely that it remains the case.

4.2 BSD

x86-based BSD systems are also best served by using Memdump. For all x86-based systems encountered through experimentation in this work, memory acquisition occurred without issue and was able to fully acquire all memory for each operating system: FreeBSD, NetBSD and OpenBSD, up to each system's 32-bit memory limit (4 GiB RAM).

Where x64 BSD systems were concerned memory acquisition was less straightforward. Under x64 OpenBSD, the maximum detected and supported memory size was 4 GiB RAM, even though an x64 kernel was running. This is a known issue with the default OpenBSD x64 kernel, which incidentally is the most likely to be encountered by investigators. For this specific operating system, the memory dump conducted using Memdump was acquired without difficulty up to 4 GiB RAM. As for x64-based FreeBSD and NetBSD operating systems, Memdump will successfully acquire all the operating system's memory. With x64-based FreeBSD and OpenBSD systems, once all the operating system's memory is acquired and up to the detected memory limit, the dump file will continue filling up with binary zeroes until either the partition where the dump-file is located fills up or the investigator stops the program's execution. Extraction of memory from these dump files is done by copying out the acquired data up to each operating system's detected memory limit. This operation can be readily carried out using the UNIX *DD* command.

4.3 Linux

4.3.1 Assessment of overall tool suitability

In all, eight memory acquisition tools, drivers and LKM were examined throughout this memorandum and in TM 2012-008. These tools included (in alphabetical order) *DD*, Fmem, Helix3 Pro, LiME, Memdump, Pmem, Second Look and X-Ways Capture. Most of these tools were found to be insufficient, particularly *DD*, Helix3 Pro, Memdump, Pmem and X-Ways Capture. X-Ways Capture was the worst of the list and should be avoided at all costs.

Due to the lack of useable */dev/mem* and */dev/kmem* based pseudo devices, LKM or memory drivers that operate from kernel space must be used. However, since *DD*, Helix3 Pro, Memdump and X-Ways Capture rely on */dev/mem* or */dev/kmem* for memory acquisition, they were not to be considered in this assessment.

LKMs and memory drivers running from kernel space have full access to the same memory that the kernel does. However, getting these LKMs or memory drivers to work on the target system generally requires the use of a compiler. Obviously, compiling software invariably changes the system state, but to what extent, nobody knows. Currently, Fedora and Red Hat based systems have readily available kernel crash drivers that are loadable at any time by the root user using the *insmod crash* command. However, because the implementation of this crash driver is not uniform amongst distributions, it is best not to rely on it. As such, the assessments made by the authors herein renounce the use of such a crash driver.

4.3.2 Assessment of acquisition speed

Based on the experimental results obtained herein, Second Look appeared to be the fastest tool. However, to be fair, LiME was acquired using the Linux kernel's NTFS driver that is significantly slower than the NTFS-3 G FUSE filesystem driver. Moreover, LiME-based dumps were saved to an external USB 2.0 device mounted by the virtual machine, while for Pmem, Fmem and Second Look, memory dumps were saved to the virtual disks of the underlying virtual machines. Therefore, writing to this external device was significantly slower than writing to a virtual disk. Thus, based on these results, the authors will not draw conclusions at this time with respect to which memory acquisition tool was faster. Instead, they will emphasize on the acquisition correctness and reliability over speed.

4.3.3 Assessment of prior Fmem and Second Look experiments

The reason Fmem and Second Look memory acquisitions were not carried out against x86 and x64 based Fedora 17 and Ubuntu 12.04 systems were that the authors had no concerns about their ability to function correctly. Acquiring memory under these two operating systems went without error in TM 2012-008. However, since both Pmem and LiME were the new tools to be examined and tested in this memorandum, the onus was on them to perform up to the expectations delivered by Fmem and Second Look. The fact that Fmem memory acquisition experiments were carried out a second time in this work does not undermine the tool's capability. Instead, it demonstrates that the original experiments conducted in TM 2012-008 were incorrectly executed.

4.3.4 Assessment of tool acquisition

LiME was specifically designed to minimize its memory footprint. It does not need a memory driver in order to map memory pages into user space and dump them using tools such as *DD*.

In terms of size-based correctness as per */proc/iomem*, Fmem and Second Look delivered memory acquisitions of the correct size every time without error. LiME was a close third and when its dumps were not exactly the same size as physical memory, it was never more than one page short (4 KiB). The same cannot be said for memory images acquired using Pmem and it is to be considered altogether unacceptable for x86 PAE-based memory acquisition. However, its x86 and x64 acquisitions were generally off by one byte of memory, with the exception of Ubuntu 11.04 x86 (which was off by 58,044,416 bytes). This was clearly due to an error in Pmem's assessment of memory size.

However, in some cases, LiME came up short in certain acquisitions, and even though it is very unlikely that any data of value was left out, this issue could be argued in court.

4.3.5 Volatility memory image assessment

Only LiME-based *lime* and *padded* formatted memory acquisitions were fully analysable using Volatility 2.3 SVN r2754. Those from Second Look, Pmem and Fmem had varying degrees of success, regardless of their analysis using Volatility 2.2 or 2.3. The reason the LiME memory images were successfully analysed using Volatility, in comparison to its counterparts (Second Look, Fmem and Pmem), is that the secondary author proposed a change to Volatility 2.3, which provided some missing capability. However, making additional modifications in order to get Volatility correctly supports the other tools' memory dumps was not a realistic endeavour and was not undertaken.

4.3.6 Conclusion

Based on all the various experiments conducted by the authors, both in this memorandum and in TM 2012-008, the authors have concluded that LiME should be considered as the investigator's primary Linux-based memory acquisition tool. This conclusion is despite the fact that Second Look and Fmem yielded accurate memory acquisitions in comparison to LiME, which was very close and never more than one memory page short. This is primarily due to LiME's implementation that minimizes its memory footprint [23].

The forensic accuracy of kernel crash based drivers has not yet been proven. Although they are likely sufficient to pass forensic reliability assessments, the authors cannot make this assertion at this time, due to insufficient information and evidence supporting this position.

Thus, erring to the side of caution, the authors are recommending the use of LiME before any other Linux-based memory acquisition tool. Barring the inability for an investigator to get LiME functioning correctly, due to its need to be compiled, the authors would then feel comfortable recommending the use of Second Look as an alternative. If Second Look is to be used, the Pmad driver can be used if the investigator is able to load it into kernel space. Because it is not currently known if crash drivers are forensically sound and the supplied Pmad driver was always used in lieu of the underlying system's kernel crash driver during the tests, the authors can only recommend using Second Look's Pmad driver to acquire memory.

This page intentionally left blank.

References

[1] Halderman, J. Alex, Schoen, Seth D., Heninger, Nadia, et al. Lest We Remember: Cold Boot Attacks on Encryption Keys. Research paper. Published in Proceedings 2008 USENIX Security Symposium. February 2008. Princeton University. http://citp.princeton.edu/pub/coldboot.pdf.

[2] Carbone, Richard. An in-depth analysis of the cold boot attack: Can it be used for sound forensic memory acquisition? Technical memorandum. TM-2010-296. Defence R&D Canada – Valcartier. January 2011. http://pubs.drdc.gc.ca/PDFS/unc105/p534323_A1b.pdf.

[3] Wikipedia. Observer effect (information technology). Online encyclopaedia. Wikimedia Foundation. April 2012. http://en.wikipedia.org/wiki/Observer_effect_(information_technology).

[4] Wikipedia. Observer effect (physics). Online encyclopaedic article. Wikimedia Foundation Inc. October 2012. http://en.wikipedia.org/wiki/Observer_effect_(physics).

[5] Carbone, Richard and Charpentier, Robert. Life-Cycle Support for Information Systems Based on Free and Open Source Software. Technical paper. Document No.: I-136. Presented to: 11th International Command and Control Research and Technology Symposium, Cambridge UK. September 2006. http://www.dodccrp.org/events/11th_ICCRTS/html/papers/136.pdf.

[6] Wikipedia. Caldera OpenLinux. Online encyclopaedic article. Wikimedia Foundation Inc. September 2011. http://en.wikipedia.org/wiki/Caldera_OpenLinux.

[7] Wikipedia. Fedora (operating system). Online encyclopaedic article. Wikimedia Foundation Inc. August 2011. http://en.wikipedia.org/wiki/Fedora_(operating_system).

[8] Berkholz, Donnie. Ranking Linux distributions, and the decline of traditional distros. Redmonk. Informational web site. http://redmonk.com/dberkholz/2013/05/20/ranking-linux-distributions-and-the-decline-of-the-traditional-distros/.

[9] Wikipedia. Physical Address Extension. Online encyclopaedic article. Wikimedia Foundation Inc. July 2011. http://en.wikipedia.org/wiki/Physical_Address_Extension.

[10] Wikipedia. x64. Online encyclopaedic article. Wikimedia Foundation Inc. July 2011. http://en.wikipedia.org/wiki/x64.

[11] Levy, Jamie. JL's stuff: /dev/crash Driver. Blog. Gleeda.blogspot.com. August 2009. Gleeda.blogspot.com. http://gleeda.blogspot.com/2009/08/devcrash-driver.html.

[12] Van de Ven, Arjan. X86: introduce /dev/mem restrictions with a config option. LWN.net. Jan 2008. http://lwn.net/Articles/267427/.

[13] Lineberry, Anthony. Malicious Code Injection via /dev/mem. Technical report. Anthony Lineberry. March 2009. http://www.blackhat.com/presentations/bh-europe-09/Lineberry/BlackHat-Europe-2009-Lineberry-code-injection-via-dev-mem.pdf.

[14] Rintel, Lubomir. Bug 492803 – Please disable CONFIG_STRICT_DEVMEM. Bug report. Redhat Bugzilla. Unknown date. https://bugzilla.redhat.com/show_bug.cgi?id=492803.

[15] Engelhardt, Jan. LKML: Arjan van de Ve: Re: [PATCH] make /dev/kmem a config option. Blog. Linux Kernel Mailing List. February 2008. https://lkml.org/lkml/2008/2/10/328.

[16] Ubuntu. Security/Features – Ubuntu Wiki. Informational web site. Ubuntu. June 2011. https://wiki.ubuntu.com/Security/Features.

[17] Man Pages Project. Mem(4) Man Page. Linux/UNIX man page. Man Pages Project. November 1992.

[18] Man Pages Project. Proc(5) Man Page. Linux/UNIX man page. Man Pages Project. October 2010.

[19] Oracle Corporation. Mem(7D) Man Page. Solaris man page. Oracle Corporation. February 2002.

[20] Oracle Corporation. Man pages section 7: Device and Network Interfaces. Reference manual. Part No.: 819-2254-33. Oracle Corporation. 2010. http://download.oracle.com/docs/cd/E19082-01/819-2254/.

[21] Kernel developers. THE /proc FILESYSTEM. Linux kernel documentation. June 2009. www.kernel.org/doc/Documentation/filesystems/proc.txt.

[22] Hale, Michael. LinuxMemoryForensics: Instructions on how access and use the Linux support. Informational web site. Volatility. October 2012. http://code.google.com/p/volatility/wiki/LinuxMemoryForensics.

[23] Sylve, Joe; Case, Andrew et al. Acquisition and analysis of volatile memory from android devices. Technical paper. Journal of Digital Investigation 2012. Department of Computer Science (University of New Orleans) and Digital Forensics Solutions. 2012. http://www.memoryanalysis.net/research/android-memory-analysis-DI.pdf.

[24] Volatility Framework Team. LinuxMemoryForensics: Instructions on how access and use the Linux support. Online resource. 2012. http://code.google.com/p/volatility/wiki/LinuxMemoryForensics.

[25] Wikipedia. PCI hole. Online encyclopaedic article. Wikimedia Foundation Inc. January 2013. http://en.wikipedia.org/wiki/PCI_hole.

[26] Noon, David W. In response to - Why does code fail to find *exact* amount of RAM?? Informational web site. Mombu.com. http://www.mombu.com/programming/hpux/t-why-does-code-fail-to-find-exact-amouut-of-ram-1573802.html.

[27] Intel Corporation. Intel 64 and IA-32 Architectures Software Developer's Manual: Volume 3A: System Programming Guide, Part 1. Page 3-7. Technical reference guide. Intel Corporation. May 2011. http://download.intel.com/design/processor/manuals/253668.pdf.

[28] OSDev.org. Memory Map (x86). Online technical reference. OSDev.org. May 2012. http://wiki.osdev.org/Memory_Map_(x86).

This page intentionally left blank.

Annex A Computer systems used for experimentation

A.1 Dedicated workstation configuration for LiME acquisition (RCMP)

In order to conduct the various LiME-based experiments against an assortment of prebuilt and preconfigured VirtualBox-based virtual machines (see Annex B for details), the following computer system configuration was used.

Table A.1: Host computer configuration for LiME memory acquisitions

Computer model	MacBook Pro
Processors	Intel Core i7 2720QM @ 2,200 MHz
Physical RAM	16,384 KB RAM
Swap	C:\Pagefile.sys
Operating System	Windows Seven Professional SP1 64-bit
Virtualization Software	Oracle VirtualBox 4.2.0 r80737 with Extension Pack
Graphics adapter	AMD Radeon HD 6750M
Graphics driver	Catalyst 11.1 (Driver: v 8.812-110104a-116524C-Apple)
Monitor	LTN170CT10 Color LCD – 17”
Floppy	N/A
USB	3 USB ports
Keyboard	French Canada Multilanguage (Apple)
Mouse	USB optical mouse
FireWire	1 FireWire ports (no attached devices)
Thunderbolt	1 Thunderbolt port (no attached devices)
DVD Drive	MATSHITA DVD-R UJ-898
Hard drive	1 - TOSHIBA MK7559GSXF (750 GB)
Sound card	Cirrus Logic CS4206A
Network cards	1) Broadcom NetXtreme BCM57765 Gigabit Ethernet PCIe 2) Broadcom 802.11n Network Adapter

A.2 Dedicated workstation configuration for Pmem acquisition (DRDC)

In order to conduct the various LiME-based experiments against an assortment of prebuilt and preconfigured VirtualBox-based virtual machines (see Annex B for details), the following computer system configuration was used.

Table A.2: Host computer configuration for carrying out Pmem memory acquisitions

Computer model	Dell OptiPlex 990
Processors	Intel Core i7 2600 @ 3,400 MHz
Physical RAM	16,384 KB RAM
Swap	F:\Pagefile.sys
Operating System	Windows Seven Professional SP1 64-bit
Virtualization Software	Oracle VirtualBox 4.2.0 r80737 with Extension Pack
Graphics adapter	AMD Radeon HD 5670
Graphics driver	Catalyst 12.10
Monitor	2 BenQ 19" monitors
Floppy	N/A
USB	10 USB ports
Keyboard	U.S. International English
Mouse	USB optical mouse
FireWire	N/A
Thunderbolt	N/A
DVD Drive	Pioneer DVD-RW DVR-111D USB Device Hitachi-LG HL-DT-ST DVD+-RW GH70N
Hard drives	1 – Seagate ST95005620AS (500 GB) 1 – Seagate ST31500541AS (1.5 TB) 1 – RevoDrive PCIe SSD (120 GB)
Sound card	RealTek HD
Network card	Intel 82579LM Gigabit Ethernet PCIe

A.3 Dedicated workstation configuration for Fmem acquisition (DRDC)

In order to conduct the various LiME-based experiments against an assortment of prebuilt and preconfigured VirtualBox-based virtual machines (see Annex B for details), the following computer system configuration was used.

Table A.3: Host computer configuration for carrying out Pmem memory acquisitions

Computer model	Dell Precision 690 Workstation
Processors	Dual Xeon 3.20 GHz w/HyperThreading (8 logical processors)
Physical RAM	22.00 GiB RAM
Swap	None
Operating System	Linux Fedora Core 14, x64
Virtualization Software	Oracle VirtualBox 4.1.0 with Extension Pack
Graphics adapter	NVidia GeForce GTX 460
Graphics driver	NVidia driver 270.41.06
Monitor	1) Dell E196FP LCD display (19”) 2) BenQ FP992 LCD display (19”)
Floppy	1.44 MB floppy drive
USB	8 USB ports
Keyboard	USB US English keyboard
Mouse	USB optical mouse
FireWire	2 FireWire ports (no attached devices)
Thunderbolt	N/A
DVD Drive	Hitachi CD-RW drive Philips CD-RW/DVD-RW drive
Hard drives	1 – Seagate ST95005620AS (500 GB) 1 – Seagate ST31500541AS (1.5 TB) 1 – RevoDrive PCIe GB SSD (120 GB) 1 – Seagate 7,200 RPM SATA (1.5 TB) 3 – Hitachi 7,200 RPM SATA in RAID 5 (2 TB) 8 – Seagate 7,200 RPM SATA drive in RAID 5 (2 TB)
Sound card	Sigma Tel HD sound card

Network card	1 – Broadcom NetXtreme Gigabit Ethernet 1 – 1394 Net Adapter
Host adapters	2x Vantec PCIe E-SATA host adapters

Annex B VirtualBox and operating system configurations for Pmem and LiME

B.1 Configuration for Ubuntu 11.04 Linux

The following are the technical details for the configuration of the three Ubuntu-based operating systems (x86, x86 PAE and x64) experimented upon in this work.

Table B.1: Ubuntu 11.04 VirtualBox virtual machine configuration details

VirtualBox configuration	x86 OS	x86 PAE OS	x64 OS
VirtualBox version	4.2.0	4.2.0	4.2.0
VirtualBox VT-x, AMD-V, Nested Paging, PAE/NX enabled	Yes	Yes	Yes
VirtualBox IO APIC enabled	Yes	Yes	Yes
VirtualBox allocated memory	8,388,608 KiB	8,388,608 KiB	8,388,608 KiB
VirtualBox allocated processors	2 processors	2 processors	2 processors
VirtualBox hard disk drive size (using SATA controller)	20.00 GB	20.00 GB	20.00 GB
VirtualBox floppy drive allocated	None	None	None
VirtualBox optical drive allocated (using IDE controller)	1 CD/DVD drive	1 CD/DVD drive	1 CD/DVD drive
VirtualBox allocated monitors	1 monitor	1 monitor	1 monitor
VirtualBox allocated video memory	128 MiB	128 MiB	128 MiB
VirtualBox 3D acceleration enabled	Yes	Yes	Yes
VirtualBox 2D acceleration enabled	No	No	No
VirtualBox network adapter enabled	Intel Pro/1000 MT Desktop	Intel Pro/1000 MT Desktop	Intel Pro/1000 MT Desktop
VirtualBox sound adapter enabled	PulseAudio / ICH AC 97	PulseAudio / ICH AC 97	PulseAudio / ICH AC 97
VirtualBox serial ports enabled	No	No	No
VirtualBox USB enabled	Yes	Yes	Yes
VirtualBox USB 2.0 (EHCI) enabled	Yes	Yes	Yes

B.2 Configuration for Ubuntu 12.04 Linux

The following are the technical details for the configuration of the three Ubuntu-based operating systems (x86, x86 PAE and x64) experimented upon in this work.

Table B.2: Ubuntu 12.04 VirtualBox virtual machine configuration details

VirtualBox configuration	x86 OS	x86 PAE OS	x64 OS
VirtualBox version	4.2.0	4.2.0	4.2.0
VirtualBox VT-x, AMD-V, Nested Paging, PAE/NX enabled	Yes	Yes	Yes
VirtualBox IO APIC enabled	Yes	Yes	Yes
VirtualBox allocated memory	8,388,608 KiB	8,388,608 KiB	8,388,608 KiB
VirtualBox allocated processors	4 processors	4 processors	2 processors
VirtualBox hard disk drive size (using SATA controller)	25.00 GB	25.00 GB	25.00 GB
VirtualBox floppy drive allocated	None	None	None
VirtualBox optical drive allocated (using IDE controller)	1 CD/DVD drive	1 CD/DVD drive	1 CD/DVD drive
VirtualBox allocated monitors	1 monitor	1 monitor	1 monitor
VirtualBox allocated video memory	128 MiB	128 MiB	128 MiB
VirtualBox 3D acceleration enabled	Yes	Yes	Yes
VirtualBox 2D acceleration enabled	No	No	No
VirtualBox network adapter enabled	Intel Pro/1000 MT Desktop	Intel Pro/1000 MT Desktop	Intel Pro/1000 MT Desktop
VirtualBox sound adapter enabled	PulseAudio / ICH AC 97	PulseAudio / ICH AC 97	PulseAudio / ICH AC 97
VirtualBox serial ports enabled	No	No	No
VirtualBox USB enabled	Yes	Yes	Yes
VirtualBox USB 2.0 (EHCI) enabled	Yes	Yes	Yes

B.3 Configuration for Fedora Core 15 Linux

The following are the technical details for the configuration of the Fedora Core-based operating systems (x86 PAE and x64) experimented upon in this work.

Table B.3: Fedora Core 15 VirtualBox virtual machine configuration details

VirtualBox configuration	x86 PAE OS	x64 OS
VirtualBox version	4.2.0	4.2.0
VirtualBox VT-x, AMD-V, Nested Paging, PAE/NX enabled	Yes	Yes
VirtualBox IO APIC enabled	Yes	Yes
VirtualBox allocated memory	8,388,608 KiB	8,388,608 KiB
VirtualBox hard disk drive size (using SATA controller)	52.34 GB	52.34 GB
VirtualBox allocated processors	2 processors	2 processors
VirtualBox floppy drive allocated	None	None
VirtualBox optical drive allocated (using IDE controller)	1 CD/DVD drive	1 CD/DVD drive
VirtualBox allocated monitors	1 monitor	1 monitor
VirtualBox allocated video memory	128 MiB	128 MiB
VirtualBox 3D acceleration enabled	Yes	Yes
VirtualBox 2D acceleration enabled	No	No
VirtualBox network adapter enabled	Intel Pro/1000 MT Desktop	Intel Pro/1000 MT Desktop
VirtualBox sound adapter enabled	PulseAudio / ICH AC 97	PulseAudio / ICH AC 97
VirtualBox serial ports enabled	No	No
VirtualBox USB enabled	Yes	Yes
VirtualBox USB 2.0 (EHCI) enabled	Yes	Yes

B.4 Configurations for Fedora Core 17 Linux

The following are the technical details for the configuration of the Fedora Core-based operating systems (x86 PAE and x64) experimented upon in this work.

Table B.4: Fedora Core 17 Linux VirtualBox virtual machine configuration details

VirtualBox configuration	x86 PAE OS	x64 OS
VirtualBox version	4.2.0	4.2.0
VirtualBox VT-x, AMD-V, Nested Paging, PAE/NX enabled	Yes	Yes
VirtualBox IO APIC enabled	Yes	Yes
VirtualBox allocated memory	8,388,608 KiB	8,388,608 KiB
VirtualBox hard disk drive size (using SATA controller)	25.00 GB	25.00 GB
VirtualBox allocated processors	4 processors	4 processors
VirtualBox floppy drive allocated	None	None
VirtualBox optical drive allocated (using IDE controller)	1 CD/DVD drive	1 CD/DVD drive
VirtualBox allocated monitors	1 monitor	1 monitor
VirtualBox allocated video memory	128 MiB	128 MiB
VirtualBox 3D acceleration enabled	Yes	Yes
VirtualBox 2D acceleration enabled	No	No
VirtualBox network adapter enabled	Intel Pro/1000 MT Desktop	Intel Pro/1000 MT Desktop
VirtualBox sound adapter enabled	PulseAudio / ICH AC 97	PulseAudio / ICH AC 97
VirtualBox serial ports enabled	No	No
VirtualBox USB enabled	Yes	Yes
VirtualBox USB 2.0 (EHCI) enabled	Yes	Yes

B.5 VirtualBox guest operating system configuration

B.5.1 Configuration for Ubuntu 11.04 Linux

The following details the various detected hardware for the Ubuntu virtualized guest operating systems (x86, x86 PAE and x64) experimented upon in this work. Please note that the x86 PAE kernel is not quite the same version as the kernel used for the x86 and x64 implementation of Ubuntu, but the authors do not see this as being a potential source of interference in their results.

Table B.5: Ubuntu 11.04 Linux guest operating system details

Operating system details	**x86 OS**	**x86 PAE OS**	**x64 OS**
Operating system kernel version	2.6.38-8-generic #42-Ubuntu SMP Mon Apr 11 03:31:50 UTC 2011 i686 i686 i386 GNU/Linux	2.6.38-10-generic-pae #46-Ubuntu SMP Tue Jun 28 16:54:49 UTC 2011 i686 i686 i386 GNU/Linux	2.6.38-8-generic #42-Ubuntu SMP Mon Apr 11 03:31:24 UTC 2011 x86_64 x86_64 x86_64 GNU/Linux
Memory detected	3,613,268 KiB	8,265,044 KiB	8,194,124 KiB
Processors detected	2 processors	2 processors	2 processors
Hard drive detected and all partitions accessible	Yes	Yes	Yes
Optical drive detected and accessible	Yes	Yes	Yes
Network adapter functioning correctly	Yes	Yes	Yes
USB detected	Yes	Yes	Yes
VirtualBox Guest Additions installed	Yes	Yes	Yes
Version	4.2.0	4.2.0	4.2.0

B.5.2 Configuration for Ubuntu 12.04 Linux

The following details the various detected hardware for the Ubuntu virtualized guest operating systems (x86 and x64) experimented upon in this work.

Table B.6: Ubuntu 12.04 Linux guest operating system details

Operating system details	**x86 OS**	**x86 PAE OS**	**x64 OS**
Operating system kernel version	3.2.0-32-generic #51-Ubuntu SMP Wed Sep 26 21:32:50 UTC i686 i686 i386 GNU/Linux	3.2.0-32-generic #51-Ubuntu SMP Wed Sep 26 21:54:23 UTC i686 i686 i386 GNU/Linux	3.2.0-32-generic #51-Ubuntu SMP Wed Sep 26 21:33:09 UTC x86_64 x86_64 x86_64 GNU/Linux
Memory detected	3,616,096 KiB	8,272,916 KiB	8,178,624 KiB
Processors detected	4 processors	4 processors	4 processors
Hard drive detected and all partitions accessible	Yes	Yes	Yes
Optical drive detected and accessible	Yes	Yes	Yes
Network adapter functioning correctly	Yes	Yes	Yes
USB detected	Yes	Yes	Yes
VirtualBox Guest Additions installed	Yes	Yes	Yes
Version	4.2.0	4.2.0	4.2.0

B.5.3 Configuration for Fedora Core 15 Linux

The following details the various detected hardware for the Fedora Core virtualized guest operating systems (x86 and x64) experimented upon in this work.

Table B.7: Fedora Core 15 guest operating system details

Operating system details	x86 PAE	x64 OS
Operating system kernel version	2.6.41.10-3.fc15.i686.PAE #1 SMP Mon Jan 23 15:36:55 UTC 2012 i686 i686 i386 GNU/Linux	2.6.41.10-3.fc15.x86_64 #1 SMP Mon Jan 23 15:46:37 UTC 2012 x86_64 x86_64 x86_64 GNU/Linux
Memory detected	8,273,772 KiB	8,179,108 KiB
Processors detected	2 processors	2 processors
Hard drive detected and all partitions accessible	Yes	Yes
Optical drive detected and accessible	Yes	Yes
Network adapter functioning correctly	Yes	Yes
USB detected	Yes	Yes
VirtualBox Guest Additions installed	Yes	Yes
Version	4.2.0	4.2.0

B.5.4 Configuration for Fedora Core 17 Linux

The following details the various detected hardware for the Fedora Core virtualized guest operating systems (x86 and x64) experimented upon in this work.

Table B.8: Fedora Core 17 guest operating system details

Operating system details	x86 PAE	x64 OS
Operating system kernel version	3.6.1-1.fc17.i686.PAE #1 SMP Wed Oct 10 12:32:58 UTC 2012 i686 i686 i386 GNU/Linux	3.6.1-1.fc17.x86_64 #1 SMP Oct 10 12:13:05 UTC 2012 x86_64 x86_64 x86_64 GNU/Linux
Memory detected	8,290,544 KiB	8,178,556 KiB
Processors detected	4 processors	4 processors
Hard drive detected and all partitions accessible	Yes	Yes
Optical drive detected and accessible	Yes	Yes
Network adapter functioning correctly	Yes	Yes
USB detected	Yes	Yes
VirtualBox Guest Additions installed	Yes	Yes
Version	4.2.0	4.2.0

Annex C VirtualBox and operating system configurations for Fmem

The virtual machines used for Fmem memory acquisition are the same ones used for Fmem in TM 2012-008, i.e., Oracle VirtualBox version 4.1.0 with Expansion Pack. For additional details, consult TM 2012-008.

This page intentionally left blank.

Annex D Experimental results

D.1 LiME

In this sub-annex, the experimental results obtained using the LiME tool are examined. This tool was used against all of the various x86 and x64 Linux operating systems examined herein.

D.1.1 Ubuntu Linux 11.04

Table D.1: Memory dump results for Ubuntu 11.04 Linux x86 using LiME

Virtual machine allocated RAM	8,388,608 KiB (8,192 MiB)
Detected memory (*cat /proc/meminfo | grep MemTotal*)	3,613,268 KiB (3,528.58 MiB)
RAM memory addresses (*cat /proc/iomem | grep "System RAM"*)	00010000 - 0009FBFF 00100000 - DFFEFFFF
Total system memory, last address and memory ranges as per */proc/iomem*	00010000 - 0009FBFF + 1 = 588,800 00100000 - DFFEFFFF + 1 = 3,756,982,272 **Last address in memory range = 3,758,030,848** **Size of System RAM = 3,757,571,072 bytes (3583.50 MiB)**
Does tool perform hash verification?	No
Is root access needed?	Yes
Raw dump	
Time (Last Write Time - Create Time)	00:11:51
Dump file size	3,757,571,072 bytes (3,583.50 MiB)
Did Volatility *linux_pslist* command succeed?	No
Padded dump	
Time (Last Write Time - Create Time)	00:15:44
Dump file size	3,758,030,848 bytes (3,583.94 MiB)
Did Volatility *linux_pslist* command succeed?	Yes

Lime dump	
Time (Last Write Time - Create Time)	00:17:08
Dump file size	3,757,571,136 bytes (includes 2x 32-byte headers) (3,583.50 MiB)
Did Volatility *linux_pslist* command succeed?	Yes
Notes	The LiME module compiled, loaded and dumped memory without incident. Memory dump sizes were as expected. Volatility 2.3 SVN-based memory analysis was successful for the *padded* and *lime* memory dumps. However, the *raw* memory dump could not be assessed using Volatility.

Table D.2: Memory dump results for Ubuntu 11.04 Linux x86 PAE using LiME

Virtual machine allocated RAM	8,388,608 KiB (8,192 MiB)
Detected memory (*cat /proc/meminfo | grep MemTotal*)	8,265,044 KiB (8,071.33 MiB)
RAM memory addresses (*cat /proc/iomem | grep "System RAM"*)	00010000 - 0009FBFF 00100000 - DFFEFFFF 100000000 - 21FFFFFFF
Total system memory, last address and memory ranges as per */proc/iomem*	00010000 - 0009FBFF + 1 = 588,800 00100000 - DFFEFFFF + 1 = 3,756,982,272 100000000 - 21FFFFFFF +1 = 4,831,838,208 **Last address in memory range = 9,126,805,504** **Size of System RAM = 8,589,409,280 bytes (8,191.50 MiB)**
Does tool perform hash verification?	No
Is root access needed?	Yes
Raw dump	
Time (Last Write Time - Create Time)	00:34:18
Dump file size	8,589,405,184 bytes (8,191.50 MiB)

Did Volatility *linux_pslist* command succeed?	No
Padded dump	
Time (Last Write Time - Create Time)	00:44:41
Dump file size	9,126,801,408 bytes (8,704 MiB)
Did Volatility *linux_pslist* command succeed?	Yes
Lime dump	
Time (Last Write Time - Create Time)	00:42:48
Dump file size	8,589,405,280 bytes (includes 3x 32-byte headers) (8,191.50 MiB)
Did Volatility *linux_pslist* command succeed?	Yes
Notes	The LiME module compiled and loaded without incident, but failed to dump all the system's memory (one memory page is missing). This appears to be a PAE-related memory driver issue. Volatility 2.3 SVN-based memory analysis was successful for the *padded* and *lime* memory dumps. However, the *raw* memory dump could not be assessed using Volatility.

Table D.3: Memory dump results for Ubuntu 11.04 Linux x64 using LiME

Virtual machine allocated RAM	8,388,608 KiB (8,192 MiB)
Detected memory (*cat /proc/meminfo | grep MemTotal*)	8,194,124 KiB (8,002.07 MiB)
RAM memory addresses (*cat /proc/iomem | grep "System RAM"*)	00010000 - 0009FBFF 00100000 - DFFEFFFF 100000000 - 21FFFFFFF

Total system memory, last address and memory ranges as per */proc/iomem*	00010000 - 0009FBFF + 1 = 588,800 00100000 - DFFEFFFF + 1 = 3,756,982,272 100000000 - 21FFFFFFF +1 = 4,831,838,208 **Last address in memory range = 9,126,805,504** **Size of System RAM = 8,589,409,280 bytes (8191.50 MiB)**
Does tool perform hash verification?	No
Is root access needed?	Yes
Raw dump	
Time (Last Write Time - Create Time)	00:20:14
Dump file size	8,589,409,280 bytes (8,191.50 MiB)
Did Volatility *linux_pslist* command succeed?	No
Padded dump	
Time (Last Write Time - Create Time)	00:29:01
Dump file size	9,126,805,504 bytes (exactly 8,704 MiB)
Did Volatility *linux_pslist* command succeed?	Yes
Lime dump	
Time (Last Write Time - Create Time)	00:31:04
Dump file size	8,589,409,376 bytes (includes 3x 32-byte headers) (8,191.50 MiB)
Did Volatility *linux_pslist* command succeed?	Yes
Notes	The LiME module compiled, loaded and dumped memory without incident. Memory dump sizes were as expected. Volatility 2.3 SVN-based memory analysis was successful for the *padded* and *lime* memory dumps. However, the *raw* memory dump could not be assessed using Volatility.

D.1.2 Ubuntu Linux 12.04

Table D.4: Memory dump results for Ubuntu 12.04 Linux x86 using LiME

Virtual machine allocated RAM	8,388,608 KiB (8,192 MiB)
Detected memory (*cat /proc/meminfo | grep MemTotal*)	3,616,096 KiB (3,531.34 MiB)
RAM memory addresses (*cat /proc/iomem | grep "System RAM"*)	00010000 - 0009FBFF 00100000 - DFFEFFFF
Total system memory, last address and memory ranges as per */proc/iomem*	00010000 - 0009FBFF + 1 = 588,800 00100000 - DFFEFFFF + 1 = 3,756,982,272 **Last address in memory range = 3,758,030,848** **Size of System RAM = 3,757,571,072 bytes (3,583.50 MiB)**
Does tool perform hash verification?	No
Is root access needed?	Yes
Raw dump	
Time (Last Write Time - Create Time)	00:11:56
Dump file size	3,757,571,072 bytes (3,583.50 MiB)
Did Volatility *linux_pslist* command succeed?	No
Padded dump	
Time (Last Write Time - Create Time)	00:14:51
Dump file size	3,758,030,848 bytes (3,583.94 MiB)
Did Volatility *linux_pslist* command succeed?	Yes
Lime dump	
Time (Last Write Time - Create Time)	00:15:15
Dump file size	3,757,571,136 bytes (includes 2x 32-byte headers) (3,583.50 MiB)
Did Volatility *linux_pslist* command succeed?	Yes

Notes	The LiME module compiled, loaded and dumped memory without incident. Memory dump sizes were as expected. Volatility 2.3 SVN-based memory analysis was successful for the *padded* and *lime* memory dumps. However, the *raw* memory dump could not be assessed using Volatility.

Table D.5: Memory dump results for Ubuntu 12.04 Linux x86 PAE using LiME

Virtual machine allocated RAM	8,388,608 KiB (8,192 MiB)
Detected memory (*cat /proc/meminfo | grep MemTotal*)	8,272,916 KiB (8,079.02 MiB)
RAM memory addresses (*cat /proc/iomem | grep "System RAM"*)	00010000 - 0009FBFF 00100000 - DFFEFFFF 100000000 - 21FFFFFFF
Total system memory, last address and memory ranges as per */proc/iomem*	00010000 - 0009FBFF + 1 = 588,800 00100000 - DFFEFFFF + 1 = 3,756,982,272 100000000 - 21FFFFFFF +1 = 4,831,838,208 **Last address in memory range = 9,126,805,504** **Size of System RAM = 8,589,409,280 bytes (8191.50 MiB)**
Does tool perform hash verification?	No
Is root access needed?	Yes
Raw dump	
Time (Last Write Time - Create Time)	00:32:00
Dump file size	8,589,405,184 bytes (8,191.50 MiB)
Did Volatility *linux_pslist* command succeed?	No
Padded dump	
Time (Last Write Time - Create Time)	00:39:40
Dump file size	9,126,801,408 bytes (8,704 MiB)
Did Volatility *linux_pslist* command succeed?	Yes

Lime dump	
Time (Last Write Time - Create Time)	00:41:18
Dump file size	8,589,405,280 bytes (includes 3x 32-byte headers) (8,191.50 MiB)
Did Volatility *linux_pslist* command succeed?	Yes
Notes	The Lime module compiled, loaded without incident but failed to dump all the system's memory (one memory page is missing). This appears to be a PAE-related memory driver issue. Volatility 2.3 SVN-based memory analysis was successful for the *padded* and *lime* memory dumps. However, the *raw* memory dump could not be assessed using Volatility.

Table D.6: Memory dump results for Ubuntu 12.04 Linux x64 using LiME

Virtual machine allocated RAM	8,388,608 KiB (8,192 MiB)
Detected memory (*cat /proc/meminfo | grep MemTotal*)	8,178,624 KiB (7,986.94 MiB)
RAM memory addresses (*cat /proc/iomem | grep "System RAM"*)	00010000 - 0009FBFF 00100000 - DFFEFFFF 100000000 - 21FFFFFFF
Total system memory, last address and memory ranges as per */proc/iomem*	00010000 - 0009FBFF + 1 = 588,800 00100000 - DFFEFFFF + 1 = 3,756,982,272 100000000 - 21FFFFFFF +1 = 4,831,838,208 **Last address in memory range = 9,126,805,504** **Size of System RAM = 8,589,409,280 bytes (8,191.50 MiB)**
Does tool perform hash verification?	No
Is root access needed?	Yes
Raw dump	
Time (Last Write Time - Create Time)	00:20:58
Dump file size	8,589,409,280 bytes (8,191.50 MiB)

Did Volatility linux_pslist command succeed?	No
Padded dump	
Time (Last Write Time - Create Time)	00:30:43
Dump file size	9,126,805,504 bytes (exactly 8,704 MiB)
Did Volatility *linux_pslist* command succeed?	Yes
Lime dump	
Time (Last Write Time - Create Time)	00:30:36
Dump file size	8,589,409,376 bytes (includes 3x 32-byte headers) (8,191.50 MiB)
Did Volatility *linux_pslist* command succeed?	Yes
Notes	The LiME module compiled, loaded and dumped memory without incident. Memory dump sizes were as expected. Volatility 2.3 SVN-based memory analysis was successful for the *padded* and *lime* memory dumps. However, the *raw* memory dump could not be assessed using Volatility.

D.1.3 Fedora Core 15 Linux

Table D.7: Memory dump results for Fedora Core 15 Linux x86 PAE using LiME

Virtual machine allocated RAM	8,388,608 KiB (8,192 MiB)
Detected memory (*cat /proc/meminfo | grep MemTotal*)	8,266,212 KiB (8,072.47 MiB)
RAM memory addresses (*cat /proc/iomem | grep "System RAM"*)	00010000 - 0009FBFF 00100000 - DFFEFFFF 100000000 - 21FFFFFFF

Total system memory, last address and memory ranges as per */proc/iomem*	00010000 - 0009FBFF + 1 = 588,800 00100000 - DFFEFFFF + 1 = 3,756,982,272 100000000 - 21FFFFFFF +1 = 4,831,838,208 **Last address in memory range = 9,126,805,504** **Size of System RAM = 8,589,409,280 bytes (8,191.50 MiB)**
Does tool perform hash verification?	No
Is root access needed?	Yes
Raw dump	
Time (Last Write Time - Create Time)	00:33:45
Dump file size	8,589,405,184 bytes (8,191.50 MiB)
Did Volatility *linux_pslist* command succeed?	No
Padded dump	
Time (Last Write Time - Create Time)	00:44:20
Dump file size	9,126,801,408 bytes (8,704 MiB)
Did Volatility *linux_pslist* command succeed?	Yes
Lime dump	
Time (Last Write Time - Create Time)	00:42:47
Dump file size	8,589,405,280 bytes (includes 3x 32-byte headers) (8,191.50 MiB)
Did Volatility *linux_pslist* command succeed?	Yes
Notes	The Lime module compiled, loaded without incident but failed to dump all the system's memory (one memory page is missing). This appears to be a PAE-related memory driver issue. Volatility 2.3 SVN-based memory analysis was successful for the *padded* and *lime* memory dumps. However, the *raw* memory dump could not be assessed using Volatility.

Table D.8: Memory dump results for Fedora Core 15 Linux x64 using LiME

Virtual machine allocated RAM	8,388,608 KiB (8,192 MiB)
Detected memory (*cat /proc/meminfo | grep MemTotal*)	8,194,780 KiB (8,002.71 MiB)
RAM memory addresses (*cat /proc/iomem | grep "System RAM"*)	00010000 - 0009FBFF 00100000 - DFFEFFFF 100000000 - 21FFFFFFF
Total system memory, last address and memory ranges as per */proc/iomem*	00010000 - 0009FBFF + 1 = 588,800 00100000 - DFFEFFFF + 1 = 3,756,982,272 100000000 - 21FFFFFFF +1 = 4,831,838,208 **Last address in memory range = 9,126,805,504** **Size of System RAM = 8,589,409,280 bytes (8,191.50 MiB)**
Does tool perform hash verification?	No
Is root access needed?	Yes
Raw dump	
Time (Last Write Time - Create Time)	00:18:15
Dump file size	8,589,409,280 bytes (8,191.50 MiB)
Did Volatility *linux_pslist* command succeed?	No
Padded dump	
Time (Last Write Time - Create Time)	00:25:29
Dump file size	9,126,805,504 bytes (exactly 8,704 MiB)
Did Volatility *linux_pslist* command succeed?	Yes
Lime dump	
Time (Last Write Time - Create Time)	00:26:09
Dump file size	8,589,409,376 bytes (includes 3x 32-byte headers) (8,191.50 MiB)
Did Volatility *linux_pslist* command succeed?	Yes

Notes	The LiME module compiled, loaded and dumped memory without incident. Memory dump sizes were as expected. Volatility 2.3 SVN-based memory analysis (with the modified *dwarf.py* file) was successful for the *padded* and *lime* memory dumps. However, the *raw* memory dump could not be assessed using Volatility.

D.1.4 Fedora Core 17 Linux

Table D.9: Memory dump results for Fedora Core 17 Linux x86 PAE using LiME

Virtual machine allocated RAM	8,388,608 KiB (8,192 MiB)
Detected memory (*cat /proc/meminfo | grep MemTotal*)	8,290,544 KiB (8,096.23 MiB)
RAM memory addresses (*cat /proc/iomem | grep "System RAM"*)	00010000 - 0009FBFF 00100000 - DFFEFFFF 100000000 - 21FFFFFFF
Total system memory, last address and memory ranges as per */proc/iomem*	00010000 - 0009FBFF + 1 = 588,800 00100000 - DFFEFFFF + 1 = 3,756,982,272 100000000 - 21FFFFFFF +1 = 4,831,838,208 **Last address in memory range = 9,126,805,504** **Size of System RAM = 8,589,409,280 bytes (8191.50 MiB)**
Does tool perform hash verification?	No
Is root access needed?	Yes
Raw dump	
Time (Last Write Time - Create Time)	00:34:16
Dump file size	8,589,405,184 bytes (8,191.50 MiB)
Did Volatility *linux_pslist* command succeed?	No
Padded dump	
Time (Last Write Time - Create Time)	00:45:07
Dump file size	9,126,801,408 bytes (8,704 MiB)

Did Volatility *linux_pslist* command succeed?	Yes
Lime dump	
Time (Last Write Time - Create Time)	00:34:29
Dump file size	8,589,405,280 bytes (includes 3x 32-byte headers) (8,191.50 MiB)
Did Volatility *linux_pslist* command succeed?	Yes
Notes	The Lime module compiled, loaded without incident, but failed to dump all the system's memory (one memory page is missing). This appears to be a PAE-related memory driver issue. Volatility 2.3 SVN-based memory analysis was successful for the *padded* and *lime* memory dumps. However, the *raw* memory dump could not be assessed using Volatility.

Table D.10: Memory dump results for Fedora Core 17 Linux x64 using LiME

Virtual machine allocated RAM	8,388,608 KiB (8,192 MiB)
Detected memory (*cat /proc/meminfo | grep MemTotal*)	8,178,556 KiB (7,986.87 MiB)
RAM memory addresses (*cat /proc/iomem | grep "System RAM"*)	00010000 - 0009FBFF 00100000 - DFFEFFFF 100000000 - 21FFFFFFF
Total system memory, last address and memory ranges as per */proc/iomem*	00010000 - 0009FBFF + 1 = 588,800 00100000 - DFFEFFFF + 1 = 3,756,982,272 100000000 - 21FFFFFFF +1 = 4,831,838,208 **Last address in memory range = 9,126,805,504** **Size of System RAM = 8,589,409,280 bytes (8,191.50 MiB)**
Does tool perform hash verification?	No
Root access needed?	Yes
Raw dump	
Time (Last Write Time - Create Time)	00:17:33

Dump file size	8,589,409,280 bytes (8,191.49 MiB)
Did Volatility linux_pslist command succeed?	No
Padded dump	
Time (Last Write Time - Create Time)	00:23:29
Dump file size	9,126,805,504 bytes (exactly 8,704 MiB)
Did Volatility *linux_pslist* command succeed?	Yes
Lime dump	
Time (Last Write Time - Create Time)	00:22:07
Dump file size	8,589,409,376 bytes (includes 3x 32-byte headers) (8,191.50 MiB)
Did Volatility *linux_pslist* command succeed?	Yes
Notes	The LiME module compiled, loaded and dumped memory without incident. Memory dump sizes were as expected. Volatility 2.3 SVN-based memory analysis was successful for the *padded* and *lime* memory dumps. However, the *raw* memory dump could not be assessed using Volatility.

D.2 Pmem

In this sub-annex, the experimental results obtained using the LiME tool are examined. This tool was used against all of the various x86 and x64 Linux operating systems examined herein.

D.2.1 Ubuntu Linux 11.04

Table D.11: Memory dump results for Ubuntu 11.04 Linux x86 using Pmem

Virtual machine allocated RAM	8,388,608 KiB (8,192 MiB)
Detected memory (*cat /proc/meminfo | grep MemTotal*)	3,613,268 KiB (3,528.58 MiB)
RAM memory addresses (*cat /proc/iomem | grep "System RAM"*)	00010000 - 0009FBFF 00100000 - DFFEFFFF
Total system memory, last address and memory ranges as per */proc/iomem*	00010000 - 0009FBFF + 1 = 588,800 00100000 - DFFEFFFF + 1 = 3,756,982,272 **Last address in memory range = 3,758,030,848** **Size of System RAM = 3,757,571,072 bytes (3583.50 MiB)**
Does tool perform hash verification?	No
Is root access needed?	Yes
Device	*/dev/pmem*
Time required to dump	00:00:49
Dump file size	3,699,986,432 (3,528.58 MiB)
Did Volatility *linux_pslist* command succeed	Yes
If no, how many strings were detected? (7-bit / 8-bit / 16-bit / 32-bit)	N/A
Notes	The Pmem kernel module compiled, loaded and dumped memory without incident. Volatility 2.2 memory analysis proceeded without error. Therefore, memory acquisition and analysis was successful for Ubuntu 11.04 x86 Linux.

Table D.12: Memory dump results for Ubuntu 11.04 Linux x86 PAE using Pmem

Virtual machine allocated RAM	8,388,608 KiB (8,192 MiB)
Detected memory (*cat /proc/meminfo | grep MemTotal*)	8,265,044 KiB (8,071.33 MiB)
RAM memory addresses (*cat /proc/iomem | grep "System RAM"*)	00010000 - 0009FBFF 00100000 - DFFEFFFF 100000000 - 21FFFFFFF
Total system memory, last address and memory ranges as per */proc/iomem*	00010000 - 0009FBFF + 1 = 588,800 00100000 - DFFEFFFF + 1 = 3,756,982,272 100000000 - 21FFFFFFF +1 = 4,831,838,208 **Last address in memory range = 9,126,805,504** **Size of System RAM = 8,589,409,280 bytes (8,191.50 MiB)**
Does tool perform hash verification?	No
Is root access needed?	Yes
Device	*/dev/pmem*
Time required to dump	00:00:05
Dump file size	536,870,911 bytes (512 MiB)
Did Volatility *linux_pslist* command succeed?	No
If no, how many strings were detected? (7-bit / 8-bit / 16-bit / 32-bit)	3,349,797 / 20,494,464 / 146,161 / 4,405
Notes	The Pmem kernel module compiled, loaded without incident, but failed to dump all the system's memory. This appears to be a PAE-related memory driver issue. Volatility 2.2 and 2.3 SVN r2574memory analysis failed using the *linux_pslist* plugin. However, string analysis indicates the memory image is highly populated and therefore, it is at least a partial memory dump. Attempts to use other plugins including *linux_netstat*, *linux_memmap*, and *linux_lsmod* all failed as well. Thus, it can be reasonably concluded that this memory image is incomplete and therefore, not entirely intact for Volatility 2.2 or 2.3 to work with.

Table D.13: *Memory dump results for Ubuntu 11.04 Linux x64 using Pmem*

Virtual machine allocated RAM	8,388,608 KiB (8,192 MiB)
Detected memory (*cat /proc/meminfo | grep MemTotal*)	8,194,124 KiB (8,002.07 MiB)
RAM memory addresses (*cat /proc/iomem | grep "System RAM"*)	00010000 - 0009FBFF 00100000 - DFFEFFFF 100000000 - 21FFFFFFF
Total system memory, last address and memory ranges as per */proc/iomem*	00010000 - 0009FBFF + 1 = 588,800 00100000 - DFFEFFFF + 1 = 3,756,982,272 100000000 - 21FFFFFFF +1 = 4,831,838,208 **Last address in memory range = 9,126,805,504** **Size of System RAM = 8,589,409,280 bytes (8191.50 MiB)**
Does tool perform hash verification?	No
Is root access needed?	Yes
Device	*/dev/pmem*
Time required to dump	00:01:51
Dump file size	9,126,805,503 bytes (8,704 MiB)
Did Volatility *linux_pslist* command succeed?	Yes
If no, how many strings were detected? (7-bit / 8-bit / 16-bit / 32-bit)	N/A
Notes	The Pmem kernel module compiled, loaded and dumped memory without incident. Volatility 2.2 memory analysis proceeded correctly and without error. Therefore, memory acquisition and analysis was successful for Ubuntu 11.04 x64 Linux.

D.2.2 Ubuntu Linux 12.04

Table D.14: *Memory dump results for Ubuntu 12.04 Linux x86 using Pmem*

Virtual machine allocated RAM	8,388,608 KiB (8,192 MiB)
Detected memory (*cat /proc/meminfo | grep MemTotal*)	3,616,096 KiB (3,531.34 MiB)

RAM memory addresses (*cat /proc/iomem | grep "System RAM"*)	00010000 - 0009FBFF 00100000 - DFFEFFFF
Total system memory, last address and memory ranges as per */proc/iomem*	00010000 - 0009FBFF + 1 = 588,800 00100000 - DFFEFFFF + 1 = 3,756,982,272 **Last address in memory range = 3,758,030,848** **Size of System RAM = 3,757,571,072 bytes (3,583.50 MiB)**
Does tool perform hash verification?	No
Is root access needed?	Yes
Device	*/dev/pmem*
Time required to dump	00:00:39
Dump file size	3,758,030,847 bytes (3,583.94 MiB)
Does Volatility *linux_pslist* command succeed?	Yes
If no, how many strings were detected? (7-bit / 8-bit / 16-bit / 32-bit)	N/A
Notes	The Pmem kernel module compiled, loaded and dumped memory without incident. Volatility 2.2 memory analysis proceeded without error. Therefore, memory acquisition and analysis were successful for Ubuntu 12.04 x86 Linux.

Table D.15: Memory dump results for Ubuntu 12.04 Linux x86 PAE using Pmem

Virtual machine allocated RAM	8,388,608 KiB (8,192 MiB)
Detected memory (*cat /proc/meminfo | grep MemTotal*)	8,272,916 KiB (8,079.02 MiB)
RAM memory addresses (*cat /proc/iomem | grep "System RAM"*)	00010000 - 0009FBFF 00100000 - DFFEFFFF 100000000 - 21FFFFFFF

Total system memory, last address and memory ranges as per */proc/iomem*	00010000 - 0009FBFF + 1 = 588,800 00100000 - DFFEFFFF + 1 = 3,756,982,272 100000000 - 21FFFFFFF +1 = 4,831,838,208 **Last address in memory range = 9,126,805,504** **Size of System RAM = 8,589,409,280 bytes (8191.50 MiB)**
Does tool perform hash verification?	No
Is root access needed?	Yes
Device	*/dev/pmem*
Time required to dump	00:00:05
Dump file size	536,870,911 bytes (512 MiB)
Did Volatility *linux_pslist* command succeed?	No
If no, how many strings were detected? (7-bit / 8-bit / 16-bit / 32-bit)	165,954 / 932,510 / 87 / 27
Notes	The Pmem kernel module compiled, loaded without incident, but failed to dump all the system's memory. This appears to be a PAE-related memory driver issue. Volatility 2.2 and 2.3 SVN r2574 memory analysis failed using *linux_pslist* plugin. However, string analysis indicates the memory image was only partially populated. Therefore, it appears to constitute a partial memory dump. Attempts to use other plugins including *linux_netstat*, *linux_memmap*, and *linux_lsmod* all failed as well. Thus, it can be reasonably concluded that this memory image is incomplete and therefore, not intact for Volatility 2.2 or 2.3 to work with.

Table D.16: Memory dump results for Ubuntu 12.04 Linux x64 using Pmem

Virtual machine allocated RAM	8,388,608 KiB (8,192 MiB)
Detected memory (*cat /proc/meminfo | grep MemTotal*)	8,178,624 KiB (7,986.94 MiB)

RAM memory addresses (*cat /proc/iomem | grep "System RAM"*)	00010000 - 0009FBFF 00100000 - DFFEFFFF 100000000 - 21FFFFFFF
Total system memory, last address and memory ranges as per */proc/iomem*	00010000 - 0009FBFF + 1 = 588,800 00100000 - DFFEFFFF + 1 = 3,756,982,272 100000000 - 21FFFFFFF +1 = 4,831,838,208 **Last address in memory range = 9,126,805,504** **Size of System RAM = 8,589,409,280 bytes (8,191.50 MiB)**
Does tool perform hash verification?	No
Is root access needed?	Yes
Device	*/dev/pmem*
Time required to dump	00:01:27
Dump file size	9,126,805,503 bytes (8,704 MiB)
Did Volatility *linux_pslist* command succeed?	Yes
If no, how many strings were detected? (7-bit / 8-bit / 16-bit / 32-bit)	N/A
Notes	The Pmem kernel module compiled, loaded and dumped memory without incident. Volatility 2.2 memory analysis proceeded without error. Therefore, memory acquisition and analysis were successful for Ubuntu 12.04 x64 Linux.

D.2.3 Fedora Core 15 Linux

Table D.17: Memory dump results for Fedora Core 15 Linux x86 PAE using Pmem

Virtual machine allocated RAM	8,388,608 KiB (8,192 MiB)
Detected memory (*cat /proc/meminfo | grep MemTotal*)	8,273,772 KiB (8,079.86 MiB)
RAM memory addresses (*cat /proc/iomem | grep "System RAM"*)	00010000 - 0009FBFF 00100000 - DFFEFFFF 100000000 - 21FFFFFFF

Total system memory, last address and memory ranges as per */proc/iomem*	00010000 - 0009FBFF + 1 = 588,800 00100000 - DFFEFFFF + 1 = 3,756,982,272 100000000 - 21FFFFFFF +1 = 4,831,838,208 **Last address in memory range = 9,126,805,504** **Size of System RAM = 8,589,409,280 bytes (8,191.50 MiB)**
Does tool perform hash verification?	No
Is root access needed?	Yes
Device	*/dev/pmem*
Time required to dump	00:00:05
Dump file size	536,870,911 bytes (512 MiB)
Did Volatility *linux_pslist* command succeed?	No
If no, how many strings were detected? (7-bit / 8-bit / 16-bit / 32-bit)	179,243 / 1,223,123 / 61 / 16
Notes	The Pmem kernel module compiled, loaded without incident, but failed to dump all the system's memory. This appears to be a PAE-related memory driver issue. Volatility 2.2 and 2.3 SVN r2574 memory analysis failed using *linux_pslist* plugin. However, string analysis indicates the memory image was only partially populated. Therefore, it constitutes a partial memory dump. Attempts to use other plugins including *linux_netstat*, *linux_memmap*, and *linux_lsmod* all failed as well. Thus, it can be reasonably concluded that this memory image is incomplete and therefore, not intact for Volatility 2.2 or 2.3 to work with.

Table D.18: Memory dump results for Fedora Core 15 Linux x64 using Pmem

Virtual machine allocated RAM	8,388,608 KiB (8,192 MiB)
Detected memory (*cat /proc/meminfo | grep MemTotal*)	8,179,108 KiB (7,987.41 MiB)

RAM memory addresses (*cat /proc/iomem | grep "System RAM"*)	00010000 - 0009FBFF 00100000 - DFFEFFFF 100000000 - 21FFFFFFF
Total system memory, last address and memory ranges as per */proc/iomem*	00010000 - 0009FBFF + 1 = 588,800 00100000 - DFFEFFFF + 1 = 3,756,982,272 100000000 - 21FFFFFFF +1 = 4,831,838,208 **Last address in memory range = 9,126,805,504** **Size of System RAM = 8,589,409,280 bytes (8,191.50 MiB)**
Does tool perform hash verification?	No
Is root access needed?	Yes
Device	*/dev/pmem*
Time required to dump	00:01:06
Dump file size	9,126,805,503 bytes (8,704 MiB)
Did Volatility *linux_pslist* command succeed?	No
If no, how many strings were detected? (7-bit / 8-bit / 16-bit / 32-bit)	4,926,699 / 389,941,195 / 86,605 / 3,521
Notes	The Pmem kernel module succeeded in compiling, but reported multiple errors "*ERROR: Attribute 56 (DW_AT_data_member_location)* …".[20] Once compiled, the module was able to load and dump memory without incident. The origin of this error is examined in detail in Section 1.5.2.1. It was determined that Volatility 2.2 and 2.3 SVN r2574 do not work with Fedora 15 x64. The *linux_pslist*, *linux_cpuinfo*, *linux_netstat*, *linux_memmap*, *linux_lsmod* were used to no avail. However, based on the number of strings extracted from the memory image, it can be concluded that it was intact, but there is a Volatility-specific issue with respect to Fedora memory image support.

[20] For more information, see Section 1.5.2 for details concerning compilation.

D.2.4 Fedora Core 17 Linux

Table D.19: Memory dump results for Fedora Core 17 Linux x86 PAE using Pmem

Virtual machine allocated RAM	8,388,608 KiB (8,192 MiB)
Detected memory (*cat /proc/meminfo | grep MemTotal*)	8,290,544 KiB (8,096.23 MiB)
RAM memory addresses (*cat /proc/iomem | grep "System RAM"*)	00010000 - 0009FBFF 00100000 - DFFEFFFF 100000000 - 21FFFFFFF
Total system memory, last address and memory ranges as per */proc/iomem*	00010000 - 0009FBFF + 1 = 588,800 00100000 - DFFEFFFF + 1 = 3,756,982,272 100000000 - 21FFFFFFF +1 = 4,831,838,208 **Last address in memory range = 9,126,805,504** **Size of System RAM = 8,589,409,280 bytes (8191.50 MiB)**
Does tool perform hash verification?	No
Is root access needed?	Yes
Device	*/dev/pmem*
Time required to dump	00:00:05
Dump file size	536,870,911 bytes (512 MiB)
Did Volatility *linux_pslist* command succeed?	No
If no, how many strings were detected? (7-bit / 8-bit / 16-bit / 32-bit)	151,836 / 795,968 / 58 / 21
Notes	The Pmem kernel module compiled, loaded without incident, but failed to dump all the system's memory. This appears to be a PAE-related memory driver issue. Volatility 2.2 and 2.3 SVN r2574 memory analyses failed using the *linux_pslist* plugin. However, string analysis indicates that the memory image was only partially populated. Therefore, it constitutes a partial memory dump. Attempts to use other plugins, including *linux_netstat*, *linux_memmap*, and *linux_lsmod* failed as well. Thus, it can be reasonably concluded that this memory image is incomplete

	and therefore, not intact for Volatility 2.2 or 2.3 to work with.

Table D.20: Memory dump results for Fedora Core 17 Linux x64 using Pmem

Virtual machine allocated RAM	8,388,608 KiB (8,192 MiB)
Detected memory (*cat /proc/meminfo | grep MemTotal*)	8,178,556 KiB (7,986.87 MiB)
RAM memory addresses (*cat /proc/iomem | grep "System RAM"*)	00010000 - 0009FBFF 00100000 - DFFEFFFF 100000000 - 21FFFFFFF
Total system memory, last address and memory ranges as per */proc/iomem*	00010000 - 0009FBFF + 1 = 588,800 00100000 - DFFEFFFF + 1 = 3,756,982,272 100000000 - 21FFFFFFF +1 = 4,831,838,208 **Last address in memory range = 9,126,805,504** **Size of System RAM = 8,589,409,280 bytes (8,191.50 MiB)**
Does tool perform hash verification?	No
Is root access needed?	Yes
Device	*/dev/pmem*
Time required to dump	00:01:27
Dump file size	9,126,805,503 bytes (8,704 MiB)
Did Volatility *linux_pslist* command succeed?	No. The command did not terminate after more than 10 hours of processing.
If no, how many strings were detected? (7-bit / 8-bit / 16-bit / 32-bit)	4,774,419 / 30,682,552 / 150,834 / 7,112
Notes	The Pmem kernel module compiled, loaded and dumped memory without incident. After more than 10 hours of processing using the *linux_pslist* Volatility 2.2 and 2.3 SVN r2574 plugin, it was determined that the plugin had not yet completed processing the memory image. It is likely that this is a Volatility-based issue, specific either to this version of Fedora, or to the incorrect kernel profile generation (see Section 2.1.3 for details). However, upon using the *linux_cpuinfo* plugin,

	all 4 virtual processors were seen by the plugin, indicating that the memory image appeared to be partially intact. Additional plugins were run against the memory image, including *linux_netstat*, *linux_memmap*, *linux_lsmod* and several others, all of which succeeded under Volatility 2.2 and 2.3 SVN r2574. However, based on the number of strings extracted from the memory image, it can be concluded that the memory image is intact, but that there is a Volatility-specific issue with respect to Fedora memory image support.

D.3 Fmem

In this sub-annex, the experimental results obtained using the LiME tool are examined. This tool was used against all of the various x86 and x64 Linux operating systems examined herein.

D.3.1 Ubuntu Linux 11.04

Table D.21: Memory dump results for Ubuntu 11.04 Linux x86 using Fmem

Virtual machine allocated RAM	8,388,608 KiB (8,192 MiB)
Detected memory (*cat /proc/meminfo | grep MemTotal*)	3,613,268 KiB (3,528.58 MiB)
RAM memory addresses (*cat /proc/iomem | grep "System RAM"*)	00010000 - 0009FBFF 00100000 - DFFEFFFF
Total system memory, last address and memory ranges as per */proc/iomem*	00010000 - 0009FBFF + 1 = 588,800 00100000 - DFFEFFFF + 1 = 3,756,982,272 **Last address in memory range = 3,758,030,848** **Size of System RAM = 3,757,571,072 bytes (3583.50 MiB)**
Does tool perform hash verification?	No
Is root access needed?	Yes
Device	*/dev/fmem*
Time required to dump	00:00:11
Dump file size	3,758,030,848 bytes (3,583.94 MiB)
Did Volatility *linux_pslist* command succeed?	Yes
If no, how many strings were detected? (7-bit / 8-bit / 16-bit / 32-bit)	N/A
Notes	Acquisition occurred without error or warning. Moreover, it was very fast, taking only 11 seconds for the dump. Analysis using Volatility 2.2 completed without error and was able to provide a full detailed process listing. It can be concluded that this memory acquisition was successful.

Table D.22: *Memory dump results for Ubuntu 11.04 Linux x86 PAE using Fmem*

Virtual machine allocated RAM	8,388,608 KiB (8,192 MiB)
Detected memory (*cat /proc/meminfo | grep MemTotal*)	8,265,044 KiB (8,071.33 MiB)
RAM memory addresses (*cat /proc/iomem | grep "System RAM"*)	00010000 - 0009FBFF 00100000 - DFFEFFFF 100000000 - 21FFFFFFF
Total system memory, last address and memory ranges as per */proc/iomem*	00010000 - 0009FBFF + 1 = 588,800 00100000 - DFFEFFFF + 1 = 3,756,982,272 100000000 - 21FFFFFFF +1 = 4,831,838,208 **Last address in memory range = 9,126,805,504** **Size of System RAM = 8,589,409,280 bytes (8,191.50 MiB)**
Does tool perform hash verification?	No
Is root access needed?	Yes
Device	*/dev/fmem*
Time required to dump	00:01:28
Dump file size	9,126,805,504 bytes (8,704 MiB)
Did Volatility *linux_pslist* command succeed?	Yes
If no, how many strings were detected? (7-bit / 8-bit / 16-bit / 32-bit)	N/A
Notes	Acquisition occurred without error or warning. Moreover, it was relatively fast, taking only 88 seconds for the dump. Analysis using Volatility 2.2 completed without error and was able to provide a full detailed process listing. It can be concluded that this memory acquisition was successful.

Table D.23: *Memory dump results for Ubuntu 11.04 Linux x64 using Fmem*

Virtual machine allocated RAM	8,388,608 KiB (8,192 MiB)
Detected memory (*cat /proc/meminfo | grep MemTotal*)	8,194,124 KiB (8,002.07 MiB)

RAM memory addresses (*cat /proc/iomem | grep "System RAM"*)	00010000 - 0009FBFF 00100000 - DFFEFFFF 100000000 - 21FFFFFFF
Total system memory, last address and memory ranges as per */proc/iomem*	00010000 - 0009FBFF + 1 = 588,800 00100000 - DFFEFFFF + 1 = 3,756,982,272 100000000 - 21FFFFFFF +1 = 4,831,838,208 **Last address in memory range = 9,126,805,504** **Size of System RAM = 8,589,409,280 bytes (8191.50 MiB)**
Does tool perform hash verification?	No
Is root access needed?	Yes
Device	*/dev/fmem*
Time required to dump	00:01:06
Dump file size	9,126,805,504 bytes
Did Volatility *linux_pslist* command succeed?	Yes
If no, how many strings were detected? (7-bit / 8-bit / 16-bit / 32-bit)	N/A
Notes	Acquisition occurred without error or warning. Moreover, it was relatively fast, taking only 66 seconds for the dump. Analysis using Volatility 2.2 completed without error and was able to provide a full detailed process listing. It can be concluded that this memory acquisition was successful.

D.3.2 Fedora Core 15 Linux

Table D.24: Memory dump results for Fedora Core 15 Linux x86 PAE using Fmem

Virtual machine allocated RAM	8,388,608 KiB (8,192 MiB)
Detected memory (*cat /proc/meminfo | grep MemTotal*)	8,273,772 KiB (8,079.86 MiB)
RAM memory addresses (*cat /proc/iomem | grep "System RAM"*)	00010000 - 0009FBFF 00100000 - DFFEFFFF 100000000 - 21FFFFFFF

Total system memory, last address and memory ranges as per */proc/iomem*	00010000 - 0009FBFF + 1 = 588,800 00100000 - DFFEFFFF + 1 = 3,756,982,272 100000000 - 21FFFFFFF +1 = 4,831,838,208 **Last address in memory range = 9,126,805,504** **Size of System RAM = 8,589,409,280 bytes (8,191.50 MiB)**
Does tool perform hash verification?	No
Is root access needed?	Yes
Device	*/dev/fmem*
Time required to dump	00:01:05
Dump file size	9,126,805,504 bytes (exactly 8,704 MiB)
Did Volatility *linux_pslist* command succeed?	No
If no, how many strings were detected? (7-bit / 8-bit / 16-bit / 32-bit)	3,667,870 / 48,728,386 / 18,443 / 2,978
Notes	The Fmem driver succeeded in dumping the full amount of system memory without error or warning. Attempts to use Volatility 2.2 and 2.3 SVN r2574 were unable to analyse the memory image. However, the memory image itself does appear to be intact. String count verification indicates that the memory image is populated with kernel and operating system data and structures.

Table D.25: Memory dump results for Fedora Core 15 Linux x64 using Fmem

Virtual machine allocated RAM	8,388,608 KiB (8,192 MiB)
Detected memory (*cat /proc/meminfo | grep MemTotal*)	8,179,108 KiB (7,987.41 MiB)
RAM memory addresses (*cat /proc/iomem | grep "System RAM"*)	00010000 - 0009FBFF 00100000 - DFFEFFFF 100000000 - 21FFFFFFF

Total system memory, last address and memory ranges as per */proc/iomem*	00010000 - 0009FBFF + 1 = 588,800 00100000 - DFFEFFFF + 1 = 3,756,982,272 100000000 - 21FFFFFFF +1 = 4,831,838,208 **Last address in memory range = 9,126,805,504** **Size of System RAM = 8,589,409,280 bytes (8,191.50 MiB)**
Does tool perform hash verification?	No
Is root access needed?	Yes
Device	*/dev/fmem*
Time required to dump	00:01:01
Dump file size	9,126,805,504 bytes (exactly 8,704 MiB)
Did Volatility *linux_pslist* command succeed?	No
If no, how many strings were detected? (7-bit / 8-bit / 16-bit / 32-bit)	8,032,777 / 75,249,462 / 370 / 724
Notes	The Fmem driver succeeded in dumping the full amount of system memory without error or warning. Attempts to use Volatility 2.2 and 2.3 SVN r2574 were unable to analyse the memory image. However, the memory image itself does appear to be intact. String count verification indicates that the memory image is populated with kernel and operating system data and structures.

D.4 Second Look

The experimental results obtained for Second Look can be found in TM 2012-008, Annex C.5. For information concerning the success or failure of using Volatility 2.2 and 2.3 SVN r2574 against the various Second Look memory images as per TM 2012-008, consult Annex G.1.4, where a detailed table summarizes these results.

Annex E Corrections and clarifications to TM 2012-008

This specific annex examines certain technical omissions and corrections as they pertain to TM 2012-008. They have been provided here in the interest of clearing up specific oversights made by the primary author when redacting the aforementioned memorandum.

E.1 Second Look memory acquisition specifics

In TM 2012-008, the specifics of how memory acquisition was conducted with respect to Second Look were left vague. Thus, the primary author of this technical memorandum (and the sole author of TM 2012-008) has undertaken the task of correcting certain omissions.

Specifically, the Second Look memory acquisition script, *secondlook-memdump.sh*, by default, attempts to load the running kernel's crash driver into kernel space. However, not all Linux distribution kernels are compiled with one (see Section 1.6 for details). The script loads the Linux kernel crash driver by running command *modprobe crash* and upon its successful loading, proceeds with memory acquisition. However, in the event that the script is unable to load the crash driver as it may been missing or damaged, the investigator can compile the Second Look supplied crash driver. The supplied crash[21] driver, *pmad.c*, is compiled by calling the *make* program from within the same directory as the driver's source code, which will compile the source code according the provided *Makefile*. Upon its compilation, the driver is loaded using *insmod* command and then the memory acquisition script, *secondlook-memdump.sh*, is run another time, with a user-supplied memory device, as shown below:

```
$ insmod pmad.ko

$ ./secondlook-memdump.sh  acquisition_file_name.dd  /dev/pmad
```

The supplied driver, *pmad*, once loaded into kernel space, creates kernel device */dev/pmad* from which the acquisition can be conducted against. Once the acquisition has been completed, the driver can be unloaded using the *rmmod* command.

Finally, throughout all the experimentation conducted in TM 2012-008 where Second Look was investigated, the supplied *pmad* driver was always used in lieu of the underlying system's kernel crash driver. This was done in order to ensure uniformity between the various distributions.

However, the results obtained in TM 2012-008 as they stand for Second Look continue to remain valid. They should not be considered in any way anomalous.

E.2 Fmem memory acquisition specifics

As previously stated, the memory acquisition experiments conducted using Fmem in TM 2012-008 were done using */proc/meminfo* as the basis for the upper memory address. However, based

[21] Pmad is very similar to the Fedora kernel-based crash memory driver.

on the detailed examination concerning the optimal memory dump sizes (see Section 2.2.4 for details), it has been concluded that the upper memory address-based limit should be derived from */proc/iomem*.

Thus, the Fmem-based experiments as carried out in TM 2012-008 have been redone using the same test operating systems as originally used therein. These included Fedora Core 15 x86 PAE and x64, as well as Ubuntu x86, x86 PAE and x64. The results from these newer experiments can be found in Annex D.3.

The new memory acquisition experiments carried out herein are for the most part the same as in TM 2012-008. The LKM must be compiled and inserted into memory using the *run.sh* script provided with the tool. Once loaded and using the final "System RAM" memory address obtained from */proc/iomem* and an appropriate block size, memory acquisition can commence. For example, consider a computer system equipped with exactly 8 GiB RAM:

```
$ dd if=/dev/fmem bs=1K count=8912896 of=memory_dump.dd
```

In this example, the count of 8,912,896 is the highest available "System RAM" address as per */proc/iomem*, divided by a block size (BS) of 1 KiB.

E.3 Clarification to the acquisition of hardware-reserved RAM

In TM 2012-008, the primary author referred extensively to hardware-reserved memory, also commonly known as physical memory (or RAM) set aside for use by the computer's hardware. This memory is accessible to the Linux kernel, its drivers and LKMs and is often acquired during a memory dump. However, the I/O memory specific to hardware devices (hardware buffers, cache, etc.) is dangerous to access and may result in a system crash. Neither this memorandum nor TM 2012-008 makes any effort to examine whether the physical memory inherent to a computer's hardware is acquired.

Annex F Linux-based memory acquisition tool comparison

F.1 Tool comparison

Upon examining in detail, the results obtained in Annex D for the numerous experiments conducted herein against the various Ubuntu and Fedora Core operating systems, memory acquisition tool specific tables have been prepared. These tables emphasize the difference between the tools with respect to their actual dump sizes for the various operating systems they were tested against. Moreover, since many of the dumps were not the expected size as per the last "System RAM" memory address found in */proc/iomem*, memory size differences or deltas have been provided in the following tables.

Table F.1: Second Look memory acquisition results (from TM 2012-008)

Operating system	Last address in /proc/iomem	Dump size	Delta	Header Bytes
Ubuntu 11.04 x86	3,758,030,848	3,758,030,848	0	0
Ubuntu 11.04 x86 PAE	9,126,805,504	9,126,805,504	0	0
Ubuntu 11.04 x64	9,126,805,504	9,126,805,504	0	0
Fedora 15 x86 PAE	9,126,805,504	9,126,805,504	0	0
Fedora 15 x64	9,126,805,504	9,126,805,504	0	0

Table F.2: Fmem memory acquisition results

Operating system	Last address in /proc/iomem	Dump size	Delta	Header Bytes
Ubuntu 11.04 x86	3,758,030,848	3,758,030,848	0	0
Ubuntu 11.04 x86 PAE	9,126,805,504	9,126,805,504	0	0
Ubuntu 11.04 x64	9,126,805,504	9,126,805,504	0	0
Fedora 15 x86 PAE	9,126,805,504	9,126,805,504	0	0
Fedora 15 x64	9,126,805,504	9,126,805,504	0	0

Table F.3: LiME padded format-based memory acquisition results

Operating system	Last address in /proc/iomem	Dump size	Delta	Header Bytes
Ubuntu 11.04 x86	3,758,030,848	3,758,030,848	0	0
Ubuntu 11.04 x86 PAE	9,126,805,504	9,126,801,408	-4,096	0
Ubuntu 11.04 x64	9,126,805,504	9,126,805,504	0	0
Ubuntu 12.04 x86	3,758,030,848	3,758,030,848	0	0
Ubuntu 12.04 x86 PAE	9,126,805,504	9,126,801,408	-4,096	0
Ubuntu 12.04 x64	9,126,805,504	9,126,805,280	0	0
Fedora 15 x86 PAE	9,126,805,504	9,126,801,408	-4,096	0
Fedora 15 x64	9,126,805,504	9,126,805,504	0	0
Fedora 17 x86 PAE	9,126,805,504	9,126,801,408	-4,096	0
Fedora 17 x64	9,126,805,504	9,126,805,504	0	0

Table F.4: LiME lime format-based memory acquisition results

Operating system	Size of System RAM	Dump size	Delta	Header Bytes
Ubuntu 11.04 x86	3,757,571,072	3,757,571,136	0	64
Ubuntu 11.04 x86 PAE	8,589,409,280	8,589,405,280	-4,096	96
Ubuntu 11.04 x64	8,589,409,280	8,589,409,376	0	96
Ubuntu 12.04 x86	3,757,571,072	3,757,571,136	0	64
Ubuntu 12.04 x86 PAE	8,589,409,280	8,589,405,280	-4,096	96
Ubuntu 12.04 x64	8,589,409,280	8,589,409,376	0	96
Fedora 15 x86 PAE	8,589,409,280	8,589,405,280	-4,096	96
Fedora 15 x64	8,589,409,280	8,589,409,376	0	96
Fedora 17 x86 PAE	8,589,409,280	8,589,405,280	-4,096	96
Fedora 17 x64	8,589,409,280	8,589,409,376	0	96

Table F.5: Pmem memory acquisition results

Operating system	Last address in /proc/iomem	Dump size	Delta	Header Bytes
Ubuntu 11.04 x86	3,758,030,848	3,699,986,432	-58,044,416	0
Ubuntu 11.04 x86 PAE	9,126,805,504	536,870,911	-8,589,934,593	0
Ubuntu 11.04 x64	9,126,805,504	9,126,805,503	-1	0
Ubuntu 12.04 x86	3,758,030,848	3,758,030,847	-1	0
Ubuntu 12.04 x86 PAE	9,126,805,504	536,870,911	-8,589,934,593)	0
Ubuntu 12.04 x64	9,126,805,504	9,126,805,503	-1	0
Fedora 15 x86 PAE	9,126,805,504	536,870,911	-8,589,934,593	0
Fedora 15 x64	9,126,805,504	9,126,805,503	-1	0
Fedora 17 x86 PAE	9,126,805,504	536,870,911	-8,589,934,593	0
Fedora 17 x64	9,126,805,504	9,126,805,503	-1	0

This page intentionally left blank.

Annex G Acquisition result analysis using Volatility

G.1 Analysis using Volatility

The following tables summarise the Volatility-based analyses obtained against the various memory images acquired from Pmem, LiME, Fmem and Second Look. Volatility 2.2 and 2.3 SVN r2574 were used for the various memory analyses.

G.1.1 LiME-based Volatility memory analysis

The Volatility memory analysis for the LiME-based memory acquisitions is summarised by the following table:

Table G.1: Volatility LiME padded and dump formats memory analyses

Operating system	Analysis using Volatility 2.3 SVN r2754
Ubuntu Linux 11.04 x86	Succeeded for both padded and lime dumps
Ubuntu Linux 11.04 x86 PAE	Succeeded for both padded and lime dumps
Ubuntu Linux 11.04 x64	Succeeded for both padded and lime dumps
Ubuntu Linux 12.04 x86	Succeeded for both padded and lime dumps
Ubuntu Linux 12.04 x86 PAE	Succeeded for both padded and lime dumps
Ubuntu Linux 12.04 x64	Succeeded for both padded and lime dumps
Fedora 15 x86 PAE	Succeeded for both padded and lime dumps
Fedora 15 x64	Succeeded for both padded and lime dumps
Fedora 17 x86 PAE	Succeeded for both padded and lime dumps
Fedora 17 x64	Succeeded for both padded and lime dumps

G.1.2 Pmem-based Volatility memory analysis

The Volatility memory analysis for the Pmem-based memory acquisitions is summarised by the following table:

Table G.2: Volatility Pmem memory analyses

Operating system	Analysis using Volatility 2.2	Analysis using Volatility 2.3 SVN r2754
Ubuntu Linux 11.04 x86	Succeeded	Was not required
Ubuntu Linux 11.04 x86 PAE	Failed – incomplete memory image	Failed – incomplete memory image
Ubuntu Linux 11.04 x64	Succeeded	Was not required
Ubuntu Linux 12.04 x86	Succeeded	Was not required
Ubuntu Linux 12.04 x86 PAE	Failed – incomplete memory image	Failed – incomplete memory image
Ubuntu Linux 12.04 x64	Succeeded	Was not required
Fedora 15 x86 PAE	Failed – incomplete memory image	Failed – incomplete memory image
Fedora 15 x64	Failed	Failed
Fedora 17 x86 PAE	Failed – incomplete memory image	Failed – incomplete memory image
Fedora 17 x64	Failed	Failed

G.1.3 Fmem-based Volatility memory analysis

The Volatility memory analysis for the Fmem-based memory acquisitions is summarised by the following table:

Table G.3: Volatility Fmem memory analyses

Operating system	Analysis using Volatility 2.2	Analysis using Volatility 2.3 SVN r2754
Ubuntu Linux 11.04 x86	Succeeded	Was not required
Ubuntu Linux 11.04 x86 PAE	Succeeded	Was not required
Ubuntu Linux 11.04 x64	Succeeded	Was not required
Fedora 15 x86 PAE	Failed	Failed
Fedora 15 x64	Failed	Failed

G.1.4 Second Look-based Volatility memory analysis

The Volatility memory analysis for the Second Look-based memory acquisitions is summarised by the following table:

Table G.4: Volatility Second Look memory analyses

Operating system	Analysis using Volatility 2.2	Analysis using Volatility 2.3 SVN r2754
Ubuntu Linux 11.04 x86	Succeeded	Was not required
Ubuntu Linux 11.04 x86 PAE	Succeeded	Was not required
Ubuntu Linux 11.04 x64	Succeeded	Was not required
Fedora 15 x86 PAE	Failed	Failed
Fedora 15 x64	Failed	Failed

G.2 Implications of using Volatility for Linux-based memory analysis

Based on the analyses conducted herein using Volatility 2.2 and 2.3 SVN r2574, an important implication of using Volatility stands out above the rest. Properly acquired Ubuntu-based memory images are analysable using Volatility, whereas those obtained using Fedora are not, with the exception of those obtained images using LiME (*padded* and *lime* formats). The reason for the inability of both Volatility frameworks to analyse intact Fedora-based memory dumps appears to be caused by the incorrect generation of kernel-based profiles, as examined in Section 2.1.3.

This page intentionally left blank.

Bibliography

Anderson, David. White Paper: Red Hat Crash Utility. White paper. 2008. Red Hat. http://people.redhat.com/anderson/crash_whitepaper/.

Open Source University, Red Hat Academy. Device Memory Buffers and /proc/iomem – Red Hat Academy 2.0. Informational web site. Red Hat Enterprise Linux 4 training course. Red Hat Inc. https://osu.redhat.com/content/courses/rha130-4/section_0002/tag_lessons/section_0002/section_0001/tag_resource/section_0003?set_language=en.

Oracle Corporation. Oracle VM VirtualBox User Manual. Guide. Version 4.2.0. Oracle Corporation. 2012. http://download.virtualbox.org/virtualbox/UserManual.pdf.

Wikipedia. 3 GB barrier. Online encyclopaedic article. Wikimedia Foundation Inc. December 2012. http://en.wikipedia.org/wiki/3_GB_barrier.

Wikipedia. 64-bit. Online encyclopaedic article. Wikimedia Foundation Inc. July 2011. http://en.wikipedia.org/wiki/64-bit.

Wikipedia. Fedora (operating system). Online encyclopaedic article. Wikimedia Foundation Inc. October 2012. http://en.wikipedia.org/wiki/Fedora_(operating_system).

Wikipedia. Linux. Online encyclopaedic article. Wikimedia Foundation Inc. October 2012. http://en.wikipedia.org/wiki/Linux.

Wikipedia. Physical Address Extension. Online encyclopaedic article. Wikimedia Foundation Inc. July 2011. http://en.wikipedia.org/wiki/Physical_Address_Extension.

Wikipedia. Physical Address Extension. Online encyclopaedic article. Wikimedia Foundation Inc. July 2011. http://en.wikipedia.org/wiki/Physical_Address_Extension.

Wikipedia. Red Hat Linux. Online encyclopaedic article. Wikimedia Foundation Inc. October 2012. http://en.wikipedia.org/wiki/Red_Hat_Linux.

Wikipedia. Ubuntu (operating system). Online encyclopaedic article. Wikimedia Foundation Inc. October 2012. http://en.wikipedia.org/wiki/Ubuntu_(operating_system).

Wikipedia. X86_64. Online encyclopaedic article. Wikimedia Foundation Inc. August 2011. http://en.wikipedia.org/wiki/X86_64.

List of symbols/abbreviations/acronyms/initialisms

’9X	’95, ’95A, ’95B, ’95C, ’98 and ’98SE (Second Edition)
2D	Two-Dimensional
3D	Three-Dimensional
4K	4 KiB
AMD	Advanced Micro Devices
AMD-V	Advanced Micro Devices-Virtualisation
ACPI	Advanced Configuration and Power Interface
APIC	Advanced Programmable Interrupt Controller
BDA	BIOS Data Area
BIOS	Basic Input/Output System
BSD	Berkeley Software Distribution
CD	Compact Disc
CFNOC	Canadian Forces Network Operations Centre
CORFC	Centre d'opérations des réseaux des Forces canadiennes
DND	Department of National Defence
DOS	Disk Operating System
DRDC	Defence Research & Development Canada
DVD	Digital Versatile Disc or Digital Video Disc
DVR	Digital Video Recorder
EBDA	Extended BIOS Data Area
EHCI	Enhanced Host Controller Interface
ELF	Executable and Linkable Format
FOSS	Free and Open Source Software
FTP	File Transfer Protocol
FUSE	Filesystem in USEr space
GB	Gigabyte (10^9 bytes)
GDB	GNU Debugger
GiB	Gibibyte
GICT	Groupe intégré de la criminalité technologique

GNU	GNU Not UNIX
GRC	Gendarmerie Royale du Canada
HD	High Definition
I/O or IO	Input/Output
INT	Interrupt
ITCU	Integrated Technological Crime Unit
IVT	Interrupt Vector Table
KiB	Kibibyte
LCD	Liquid Crystal Display
LiME	Linux Memory Extractor
LKM	Linux Kernel Module
LTS	Long Term Support
MHz	Megahertz
MiB	Mebibyte
N/A	Not Available
NFS	Network File System
NTFS	New Technology File System
OS	Operating System
PAE	Physical Address Extension
PAE/NX	Physical Address Extension / No eXecute
PC	Personal Computer
PCI	Peripheral Component Interconnect
PCIe	PCI (Peripheral Component Interconnect) Express
RAM	Random Access Memory
RC	Release Candidate
RCMP	Royal Canadian Mounted Police
RDDC	Recherche et développement pour la défense Canada
RPM	Red Hat Package Manager
Rsh	Remote Shell
SMP	Symmetric Multi-Processing
SP1	Service Pack 1

SPARC	Scalable Processor ARChitecture
SQ	Sûreté du Québec
SSD	Solid State Disk
Ssh	Secure Shell
TB	Terabyte
TM	Technical Memorandum
UK	United Kingdom
USB	Universal Serial Bus
UTC	Coordinated Universal Time
VT-x	Virtualisation x86
x64	Refers to the 64-bit PC architecture
x86	Refers to the 32-bit PC architecture
x86 PAE	Refers to the 32-bit PAE PC architecture

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1. ORIGINATOR (The name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Centre sponsoring a contractor's report, or tasking agency, are entered in section 8.) Defence R&D Canada – Valcartier 2459 Pie-XI Blvd North Quebec (Quebec) G3J 1X5 Canada	2. SECURITY CLASSIFICATION (Overall security classification of the document including special warning terms if applicable.) UNCLASSIFIED (NON-CONTROLLED GOODS) DMC A REVIEW: GCEC JUNE 2010

3. TITLE (The complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S, C or U) in parentheses after the title.)

The definitive guide to Linux-based live memory acquisition tools: An addendum to "State of the art concerning memory acquisition software: A detailed examination of Linux, BSD and Solaris live memory acquisition"

4. AUTHORS (last name, followed by initials – ranks, titles, etc. not to be used)

Carbone, Richard and Bourdon-Richard, Sébastien

5. DATE OF PUBLICATION (Month and year of publication of document.) September 2013	6a. NO. OF PAGES (Total containing information, including Annexes, Appendices, etc.) 122	6b. NO. OF REFS (Total cited in document.) 28

7. DESCRIPTIVE NOTES (The category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

Technical Memorandum

8. SPONSORING ACTIVITY (The name of the department project office or laboratory sponsoring the research and development – include address.)

Defence R&D Canada – Valcartier
2459 Pie-XI Blvd North
Quebec (Quebec)
G3J 1X5 Canada

9a. PROJECT OR GRANT NO. (If appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant.) 31XF20 MOU RCMP "Live Forensics"	9b. CONTRACT NO. (If appropriate, the applicable number under which the document was written.)
10a. ORIGINATOR'S DOCUMENT NUMBER (The official document number by which the document is identified by the originating activity. This number must be unique to this document.) DRDC Valcartier TM 2012-319	10b. OTHER DOCUMENT NO(s). (Any other numbers which may be assigned this document either by the originator or by the sponsor.)

11. DOCUMENT AVAILABILITY (Any limitations on further dissemination of the document, other than those imposed by security classification.)

Unlimited

12. DOCUMENT ANNOUNCEMENT (Any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in (11) is possible, a wider announcement audience may be selected.))

Unlimited

13. ABSTRACT (A brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual). (U) This technical memorandum is an addendum to TM 2012-008, “State of the art concerning memory acquisition: A detailed examination of Linux, BSD and Solaris live memory acquisition.” It examines in detail two additional software tools, Volatility’s Pmem and LiME’s Linux kernel memory drivers, both of which can be used for the memory acquisition of Linux-based live computer systems. The authors then compare them with Fmem and Second Look, the two best Linux-based memory acquisition tools as per TM 2012-008. Fmem and Second Look are analysed using the same methodology as for Pmem and LiME. This memorandum also amends information pertaining to the faulty memory acquisition of Fmem as conducted in the previous study. Additionally, certain inaccuracies were made in TM 2012-008. This specific text corrects them. As such, it should now be considered the authoritative reference concerning Linux, UNIX and BSD memory acquisition, although the experiments as conducted in TM 2012-008 will continue to remain valid. Finally, upon completing the analysis of these tools, the authors recommend the use of LiME for investigative fieldwork. However, other tool-specific recommendations are found and examined in the Conclusion. (U) Ce mémorandum technique est un addenda au TM 2012-008, “State of the art concerning memory acquisition: A detailed examination of Linux, BSD and Solaris live memory acquisition”. Il examine en détail deux outils logiciels additionnels qui peuvent être utilisés pour l’acquisition de mémoire sur des ordinateurs Linux en exécution, plus spécifiquement les pilotes de mémoire du noyau Pmem (de Volatility) et LiME. Les auteurs les comparent par la suite avec Fmem et Second Look, les deux meilleurs outils d’acquisition de mémoire sous Linux selon le TM 2012-008. Fmem et Second Look ont été analysés en utilisant la même méthodologie que pour Pmem et LiME. Ce mémorandum corrige également les informations relatives à l’acquisition de mémoire fautive de Fmem telle que menée dans l’étude précédente. De plus, certaines inexactitudes ainsi portées ont été inscrites dans le TM 2012-008. Le présent texte les corrige aussi. À ce titre, il doit être considéré comme la référence faisant autorité en ce qui concerne l’acquisition de mémoire sous Linux, UNIX et BSD, bien que les expériences menées dans le TM 2012-008 demeurent valides. Finalement, après avoir complété l’analyse de ces outils, les auteurs recommandent l’utilisation de LiME pour le travail d’enquête. Cependant, des recommandations spécifiques à d’autres outils sont aussi examinées dans la conclusion.
14.KEYWORDS, DESCRIPTORS or IDENTIFIERS (Technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus, e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.) BSD; Computer Forensics; Crash driver; DD; Digital Forensics; Fmem; FreeBSD; Helix 3 Pro; Kernel crash driver; LiME; Linux; Memdump; Memory Acquisition; NetBSD; OpenBSD; Pmem; Second Look; Solaris; UNIX; Volatility; X-Ways Capture