Title	Forensic Analysis of Virtual Hard Drives
Authors(s)	Tobin, Patrick; Le-Khac, Nhien-An; Kechadi, Tahar
Publication date	2017-03-31
Publication information	Journal of Digital Forensics, Security and Law, 12 (1): 46-58, Article 10
Publisher	The Association of Digital Forensics, Security and Law
Link to online version	https://commons.erau.edu/jdfsl/vol12/iss1/10
Item record/more information	http://hdl.handle.net/10197/9653
Publisher's version (DOI)	10.15394/jdfsl.2017.1438

Forensic Analysis of Virtual Hard Drives

***Abstract*—The issue of the volatility of virtual machines is perhaps the most pressing concern in any digital investigation involving a virtual machine. Current digital forensics tools do not fully address the complexities of data recovery that are posed by virtual hard drives. It is necessary, for this reason, to explore ways to capture evidence, other than those using current digital forensic methods. Data recovery should be done in the most efficient and secure manner, as quickly, and in an as non-intrusive way as can be achieved. All data in a virtual machine is disposed of when that virtual machine is destroyed, it may not therefore be possible to extract and preserve evidence such as incriminating images prior to destruction. Recovering that evidence, or finding some way of associating that evidence with the virtual machine before destruction of that virtual machine, is therefore crucial. In this paper we present a method for extracting evidence from a virtual hard disk drive in a quick, secure and verifiable manner, with a minimum impact on the drive thus preserving its integrity for further analysis.**

***Index Terms*—Virtual Machine, Digital Forensics, Virtual Machine Forensics**

INTRODUCTION

It is very rare to find a crime scene where a digital device of some description has not been used. Whether it is a tablet computer, a phone or perhaps portable digital storage device like a USB key or external hard drive, if they are Unix, Linux or Windows based systems, physical devices can be taken possession of for examination [1]. The data within them can be catalogued, classified, extracted and subject to detailed examination. They constitute a physical connection between the user/owner of the device, those data that are on it and how those data were used in a crime. The real, physical nature of these devices is invaluable to an investigator, but is absent where a virtual machine (VM) is involved, yet the goal of a digital examiner remains the same - to secure as much evidence as possible [2].

How evidence is collected is important to the integrity of that evidence and the subsequent conduct of any investigation. What happens to that evidence after collection is crucial, how it is saved, how it is handled and processed, and how it is related to an offence or misconduct is vitally important to an investigation. Traditional digital forensics has developed tried and tested methods of achieving these goals and tools have been developed for these purposes. Applying these to VM forensics may involve developing, or enhancing these tools or methods further, or designing new tools to take account of the absence of physical hardware.

A VM possesses all the characteristics of true hardware [3] - the virtual hard drive (vHDD) is formatted to the specifications of the operating system being used, the virtual RAM (vRAM) has all the expected attributes that true RAM has, as do the other virtual devices associated with a VM, e.g. NICs, USB controllers, graphics processors, etc. Nonetheless recovering evidence from a VM is more difficult not only because we are investigating one process of the host operating system (OS), but also because of the volatility of a VM. Evidence in a VM can be lost easily when that VM is moved[6] or deleted.

The 'throwaway' nature of VMs also allows their use as anti-forensics tools, as discussed by Barrett and Kipper in [6]. They further propose that in future a truly disposable operating system (OS) may be created for single session use, using hypervisor functions and applications moved to the Web to create that OS, and dismantled completely when shut down. This prospect will defeat any forensics tool not in a position to capture the OS and data, prior to shut down - nothing being left to analyse after the session is finished.

Cloud computing provides users with a flexibility that traditional computing lacks. It allows organisations to manage their computing needs on an on-demand basis, rather than a lead-in time of perhaps weeks or months if installing physical hardware. It allows a company to balance its workload very quickly, maintain secure images of their data, and ensure resilience against hardware failure [4]. This business model enables costs to be controlled - you pay for what you use. Cloud computing models, such as SaaS[1], DaaS, IaaS all rely on virtualisation to deliver their services [5]. These components form the basis of cloud computing, including high speed bandwidth, and distributed computing [6]. We focus on virtualisation in cloud computing. Cloud computing, and the ability to create a computing instance when required, pose Law Enforcement (LE) with a difficult investigation model. The multi-tenancy [7] nature of much of cloud computing and the sharing of resources, adds to the investigation difficulties.

In this paper we propose a method for gathering evidence from a VM's vHDD, reducing the volume of data being gathered, and minimising intrusion on a suspect VM. In the circumstances of remote acquisition of a VM's data, physical access to the hardware that a VM resides on is difficult, but will not be necessary in the context of what we propose. Our paper is organised as follows: Firstly we outline what technologies are currently available to carry out a digital forensic examination on a VM. In Section 2 we examine how to best gather data from a vHDD. We then describe our approach to VM forensics and how we implement it. Section 4 looks at how best to optimise software execution, evidence gathering, and the consequences of these for both the suspect and investigator. We support our optimisation techniques

[1]Software as a service, Desktop as a Service, Infrastructure as a Service

with metrics of execution times before and after optimisation. Finally we will conclude by outlining further research.

VIRTUAL MACHINES

VM technologies fall into two categories - Type I and Type II VMs [3]. Type I virtualisation involves a hypervisor, or virtual machine manager, (VMM) using a thin code layer to manage resources in real-time. They run directly on the hardware and are commonly known as 'bare-metal' hypervisors, examples include XenServer from Citrix, ESXi from VMware and Hyper-V from Microsoft. They reduce the overhead needed by the hypervisor itself, provide good performance, availability and security.

Type II hypervisors run as an application on top of an OS. They are very popular and are usually used to emulate another OS, e.g. running Windows within Linux, or vice versa. They are usually found on home computer systems where security and efficiency are less critical, examples include VirtualBox from Oracle and Microsoft's VirtualPC.

VM Forensics - Current State of the Art

VMs were introduced in the 1960's [8] but declined in demand due mainly to the decline in popularity of mainframes and the wider accessibility of personal computers [27]. Their recent re-emergence and use by different entities, such as public, private and individuals, has brought with it many challenges for Law Enforcement [9], and it is very likely that investigators will encounter increasing numbers of cases where VMs are used. VM digital forensics challenges are similar to those of traditional digital forensics, such as host log analysis and data capture and analysis, but recovering those data from a VM operating on a cloud platform can pose a challenge. Methods and tools exist to recover data from traditional computer systems and their hard drives, but although the methods of capturing VM data are essentially the same, collecting evidence from a vHDD is more problematic.

In a traditional digital investigation capturing the data on a hard drive usually involves capturing the suspect computer and seizing the hard drive. However, seizing the hard drive, both physical and virtual, that a VM uses is less straightforward. If the VM is operating in the cloud through a service provider, accessing the hard drive could involve removing it from the data centre, and then examining it. This is likely to take time, running the risk of data being altered, removed, deleted or destroyed, and may also reveal other users data on the hard drive, giving rise to data protection issues. There are also few tools to assist in investigating a vHDD, apart from LibVMI [15]. If the VM is operating in a desktop machine, in VirtualBox or KVM/QEMU for instance, it may be impossible to gain access to the hard drive.

VM Introspection

The most important VM forensics technology to date has been Virtual Machine Introspection [10] (VMI). VM introspection uses the virtual machine manager (VMM) to view what is happening inside a VM. It was originally introduced as a method of implementing intrusion detection systems, allowing a VM to be monitored from outside to assess what is happening inside, but it can also be used in the forensic investigation of VMs. VMI describes how a VMM administrator can inspect what is occurring inside a VM, view the VM memory, its processes, its network settings, installed OSes, applications and services. This powerful feature of VMI has allowed criminal investigation of VMs to take place and data to be captured which might otherwise have been lost.

Nance et al. [11] describes VMI as falling into two categories - those that monitor a VM and those that interfere with a VM. Using VMI to monitor the runtime state of a VM effectively allows such monitoring to take from outside the guest system, without the knowledge of that guest system [11]. Furthermore, without knowledge of VMI monitoring it is therefore neither possible to prevent it, nor possible to interfere with that monitoring [11]. Interference, on the other hand, comprises a different set of circumstances, for instance, when VMI interferes with a VM it responds to some condition in the VM that requires a response, such as a detected threat, by terminating the affected process. This interference with the guest system may alter data, and should be avoided as any change to the system being inspected could effectively alter evidence and thus possibly provide a different forensic outcome to that of an unaltered system. This will have consequences for any evidence recovered and may cause that evidence to be ruled inadmissible. VMI VMI does not affect a VM in any other way as it does not draw on any of the VM's resources.

Semantic Awareness: Th semantic gap that exists between raw data an its natural language representation, is recognised as the greatest challenge facing VM forensics. Nance et al. describe semantic awareness [11] as the VM's knowledge of its guest operating system (OS), and by Joshi et al. [28] as the level of abstraction naturally used and that exposed by a virtual machine. Bridging that gap is not a trivial process and is made more difficult by the failure of an OS being inspected to follow certain semantic expectations. However that awareness is dependent upon the OS following the known data structures and syntax of that OS. Bridging that gap is very much dependent upon the target OS following the known data structures and syntax of that OS. By failing to follow those structure and syntax's, Bahram et al. [13] described how to subvert VMI in such a way that any data recovered through VMI renders those data to be questionable. This can be achieved through the simple assumption that data on the suspect system conforms with the expected data structures and syntax of that kernel, failure to adhere to that assumption can cause those data to be suspect. This means that to

evade VMI a completely different view of the system can be presented to VMI, than that which is seen by the user. This approach can cause those data to be obfuscated in such a way that reversal of that obfuscation may be computationally very complex and very expensive, and without prior knowledge of how it is achieved, it would make tools such as The Volatility Framework of little use in analysing those subverted memory files. Compromisation can be achieved by various means, including using a rootkit, possibly causing any data recovered to be unsound, with significant implications for the value of evidence gathered from those data.

The Volatility Framework: The Volatility Framework[14] is used in forensic memory analysis. It provides an analysis platform for a wide range of file types, including core dumps, from various OSes, including Linux kernels from 2.6.11 to 4.2.3, OS X from 10.5.x to 10.11.x and most Windows OS's from Windows XP SP2 to Windows 10, and various virtual machine managers (VMMs), including VMware and VirtualBox. Linux core dumps can be dumped into ELF file format which can be parsed using Volatility. However accessing the vHDD is not possible using Volatility, as it is a memory inspection tool.

Another very useful memory acquisition and inspection tool is LibVMI [15]. This is a tool that allows reading from and writing to a VM's memory. It was developed for the Xen VMM, but has been extended to other VMMs. As Volatility was originally intended for use on static memory images the developers of LibVMI have extended its functionality to live memory address spaces by writing a Python wrapper for Volatility for use by LibVMI [15]. Although this is a powerful addition to the digital forensic examiners toolkit it is very likely to suffer a latency issue between when data are present in RAM and the when LibVMI captures them. This could cause data to be swapped out of memory, or be overwritten before LibVMI captures those data.

Best Practice Guidelines

The Association of Chief Police Officers of the UK (ACPO) [16], ISO Standard 27037 [17], U. S. Department of Justice Office of Justice Programmes National Institute of Justice [18] and the EU publication Guidelines on Digital Forensic Procedures for OLAF Staff [19] have set guidelines to be followed when examining digital evidence.

The ACPO have published four simple principles to be followed, Principles 1 and 2 are most relevant to our work. Briefly described, these are: Principle 1 expressly disallows changes to original data, Principle 2 describes how data should only be accessed by a qualified person, but allows an examiner to explain the reasons for any action taken that may have changed the original data, this second principle is important in the context of our approach to VM forensics. Principles 3 and 4 are not relevant to our work at present. These principals have been accepted as best practice by the Courts in the UK and Ireland and have influenced the drafting of the EU OLAF guidelines.

COLLECTING DATA FROM A VHDD

There are many tools available to examine data on a physical hard drive, e.g. EnCase [20], the SANS Investigative Forensics Toolkit [21], FTK [22], TSK [23], these have varying degrees of functionality. What they all have in common is that they require that the hard disk be available to be examined, or an image of that hard disk, something not necessarily possible where a VM is concerned. It is possible to obtain an image of a vHDD when a VM is captured while still live, but the volatility of VMs can still make this a difficult process. Typically VM data are captured through a snapshot of the VM, via the VMM, it preserves the VM at a specific time, but is limited in that it is a fixed image and will fail to capture data subsequent to the snapshot. The VM must also be live when taking a snapshot, rather than in the digital forensics of a standard computer where off-line capture is possible. The

ACPO Good Practice Guide for Digital Evidence and the US Department of Justice Special Report of April, 2004[18] are two very important reports and were written to contribute to a framework for ensuring gathered evidence and the methods used to recover that evidence, meet a minimum standard. They were originally intended to guide examination of standard computer systems, but these guidelines equally apply to VMs.

Data recovered from a VM can be processed in the same manner as those recovered from standard systems. In our proposal we create and recover the md5 signatures of data and propose using these signatures to match against data sets of known hash signatures of known files. For example by matching the recovered hash against those in repositories, such as the National Software Reference Library (NSRL), it is possible to identify those files where the hash signatures exist in that library. This method of file identification can be efficient, because files are identified by means of using a hash signature, reducing the data to be recovered from several MiB to 32B. We recognise that any alteration to the original data will result in a different hash signature, to that from unaltered data. This could be addressed through sub-file forensics, but we do not examine this in this research.

EVIDENCE SEARCH THROUGH INJECTED CODE

Our approach to VM forensics involves injecting forensic software into a VM and executing that software. In their paper, WITHHELD [24] described how code injection into a VM could be used to execute known benevolent code to carry out digital forensics in that VM, they elaborated on some benefits of doing this. In this paper part of this proposal is implemented and the results are described.

We have built a simple, but effective search engine for this purpose, which will have minimal impact on the host system. The purpose of this system is to search a hard drive, or

partition, for pre-defined file types using their file signature, create an md5 hash of each file found that satisfy the search criteria, and save that hash signature to a separate file for extraction by VMI software. This approach allows very fast searching of a hard drive, reduces the volume of data for extraction and minimises interference with the host system.

Evidence integrity can be compromised by writing data to a hard drive, whether virtual or physical. Preventing this in a digital forensics laboratory invariably means interfacing a write-blocker between the hard drive and the forensics tool. Using a write-blocker is not possible in the VM forensics approach we propose. To solve this problem we have written a software writ-blocker for use with this search engine. We create a small RAM disk, then install the tool into RAM disk, execute it from there and save all data found to file within that RAM disk. This prevents any data being written to the vHDD, and because the RAM disk is a reserved area of RAM no changes will occur to data in RAM. The small size of the RAM disk used, 8 MiB, has very little impact on the VM and its performance.

Our approach has some important advantages. First it significantly scales down the volume of data needed to be extracted, second it provides an investigator with a forensically sound fingerprint of a file used or distributed. Code can be tailored to suit any purpose required, it can be customised to search for and recover files, and export them for extraction by VMI software, and by using the OS semantics can help bridge the semantic gap. It can help escape kernel data structure manipulation, as described by Bahram et al. [13] by identifying the means of such manipulation.

Using an md5 signature to effectively identify file we can reduce the volume of data to 32 B per file, from a jpeg file of approximately 5 MB, a reduction in data size of approx. 1.5 x 10^4 is achieved, giving a very significant reduction in data volume to be extracted. Much more significant gains can be made with other file types, e.g. 10 TIFF files occupying 50 MB each, give a total of 500 MB of data for recovery, whereas 10 md5 signatures of those same files will occupy only 320 B. This result in extraction of a much smaller data footprint, reduce the bandwidth necessary to recover those data and minimise the risk of corruption

Providing an md5 signature of a file allows that file to be matched against databases of hash signatures of known files. The NIST National Software Reference Library (NSRL), among others, currently provide a Reference Data Set against which md5 signatures can be referenced and their corresponding files identified. This is a very fast and secure method of identifying files. Furthermore hash signatures can be used to identify files shared between users where those files are not catalogued by the NSRL, or where files are recovered from other computers and suspected to have originated from the system being inspected.

Overcoming the semantic gap is not a trivial matter, it is expensive and computationally complex. Using software injected into an OS, in the manner we describe it, and executing that software natively on the suspect machine, we are using the original data in the file system, the semantics of the target OS and the data structures of that OS. By accessing the file data present on the system we can recover those files of interest, there is no need to convert data from its raw state to its natural language representation, there is no need to address the data structures. This is a very significant advantage of our approach, as it helps reduces the time needed to examine a system, saves investigator time and reduces the data to be recovered.

We can also overcome the subversion techniques described by Bahram et al [13]. In the same way as we describe overcoming the semantic gap, we also use the kernel data structures of a compromised operating system to our advantage, by processing data inside that compromised system. We must be very cautious that by using a compromised OS we run a significant risk of compromising data recovered, but it is possible to determine the method of manipulation used and by doing so reverse that manipulation and recover the data uncompromised.

EXPERIMENT

Code Optimisation

Optimising code execution is best achieved through careful design of the algorithms used, making best use of available hardware, reducing user interaction and selective targeting of data for processing. We have used CPU affinity to make the best use of the available CPUs and CPU cache, by pinning our tools to one CPU core and timing execution.

In view of the volatility of VMs and their storage, we must avoid alerting a user to external activity on their VM as this can quickly result in the destruction of the VM and loss of all data. Therefore speed of execution of our injected software is important.

Test Environment

To build our software engine and the investigation environment we used KVM/QEMU v1.3.2 running on Sabayon Linux v15.11, kernel 4.2.0 and created a VM using the same Sabayon Linux version as the host system. We used an Intel i7 processor, at 1.7 GHz, with 4 GB RAM, and an SSD at 540 MB read speed. We used a relatively low power processor to mirror as closely as possible the performance of an Amazon T2 medium EC2 instance, to measure how our software might operate on such an instance. We gave our VM with 1,024 MB of RAM, 20 GB of SSD and 2 vCPUs. We copied a data set of 2.5 GB - 12,808 files, in 4642 directories - into the guest and used this as our test data. Allocating two vCPUs allowed us to manipulate our test platform to our own specifications. The purpose was to make comparison between two different management scenarios, one where the OS managed the vCPU allocation

and one where we pinned our program to one vCPU. We executed our program in these two management environments to find which one returned the best performance and gave the best results in terms of execution speed.

Description

To achieve our aims we built a tool to search a hard drive. The tool searches a file system for files, recursing into sub-directories. It then uses the Linux utility *file* to extract the file type, from any files found. We then *grep*'ed the file command's output to identify text files. The program then builds the full path to the files found and uses the Linux command *md5sum* to calculate the md5 hash of the files found. The 'md5sum' output is then save to file on the RAM drive.

We developed this tool on the host system described above and compiled it using the Gentoo Hardened 4.9.3 p1.1 version of gcc. We took this route building our own search engine in preference to using the Linux terminal utility '*find*'. The find command is very useful and can be tailored to a users specification by customising the path to be searched and the files to be searched for, however initial testing showed that this utility consumed a lot of CPU time, resulting in longer execution times than our own search engine when we compared those times.

The POSIX interface library contains a header file, '*ftw.h*', used to recursively search a file system tree. We wrote a program using this header file, to be used as a comparison. We used this program to make comparisons between its execution time and our custom program execution time. It was important to us to design our tool to replicate the functionalities of both find and ftw.h exactly, and we successfully did this. Our tool recursively calls directories in a file system tree, searches those directories for files appropriate to the search criteria and processes those files as required. It continues until the directory is fully searched at which point it will exit the search of that directory branch to resume its search of the parent directory.

Tool Execution

In our example we sought text files, identifying them using the Linux terminal command *file*, and generated an md5 hash for each file found. We closed all open processes prior to the test runs. We ran both programs, our search program, Tool_1, and the one using ftw.h - Tool_2 - 10 times and took the mean execution time. Initial execution times were consistently within a range that indicated that further testing of both programs would not significantly influence those results. Our VM was provisioned with two vCPUs and we carried out two separate sets of tests. In the first test run we pinned our programs to one vCPU in the VM and timed 10 runs of both tools, in the second test run we allowed the VM operating system manage CPU balancing while executing our programs. We saved the output from both sets of tests to files, and the full path to the file.

Pinning a process to one CPU, vCPU or core forces the execution of that process to be carried out exclusively on that CPU or core, CPU affinity can result in greater efficiency [25]. Efficiency can arise by optimising cache performance and reducing cache miss rates [29], task data does not need to be cycled, leading to efficiency and therefore time savings. Table 1 illustrates the results we obtained from our tools runs, we have labelled the data appropriately - pinned meaning pinned to one vCPU, unpinned meaning OS managed balancing.

Test Results

	pinned	unpinned
Tool_1	47.48s ± 0.5s	55.59s ± 1.6s
Tool_2	1 m 41s ± 2.1s/4.1s	1m 31.05s ± 2.5s/10s

Table 1. Timing of program runs of the two tools used showing ranges +/- mean.

Our test results show that our tool ran significantly faster than that using ftw.h and was faster again when CPU affinity was applied. CPU affinity had an appreciably positive outcome for execution time for our tool. An unexpected outcome of our experiments showed that there was smaller divergence from the mean execution time when our tool, Tool_1, was measured, compared with a wider divergence range when Tool_2 was tested.

We ran further tests to verify that the correct md5 hashes were being returned by our tool by taking random entries from the results files and separately calculating md5 hashes of these files. Those results confirmed to us that our tool was executing as expected. Figure 1 is an extract of the md5um output that was saved to file.

```
af9efe495c3332991faf2f49275438c6 /home/patt/.config/user-dirs.locale
```

Figure 1. Example of one md5 and the file name saved to file.

The same file with an md5 calculated from a terminal window is shown in Figure 2, with the output from the 'file' command showing that it is a text file.

```
Seoda patt # file /home/patt/.config/user-dirs.locale
/home/patt/.config/user-dirs.locale: ASCII text
Seoda patt # md5sum /home/patt/.config/user-dirs.locale
af9efe495c3332991faf2f49275438c6  /home/patt/.config/user-dirs.locale
Seoda patt #
```

Figure 2. File type and md5 signature of file shown in Figure 1 re-calculated in terminal.

The same C program wrapper and standards were used on both tools to measure execution time. Comparison of the results show that our tool runs faster than the alternative tool. In the context of our tests and the volume of data used the time differences do not appear to be of significance, but scaling to much larger file systems we would expect the disparity to be more profound and to become more apparent.

Performance and Analysis

Linux maintains a page cache to accelerate access to files. Data can very quickly be read from cache rather than re-reading the data from storage, this facility is also known as *disk buffering*[26]. This valuable feature can significantly increase the performance of processes by reading data once from disk, caching it to fast main memory and reading it from the main memory for subsequent operations involving those data, rather than accessing the very much slower disk.

In our experiments while pre-testing our tool, cached data produced very slightly anomalous results each time we timed our program operation. This occurred because we were re-using data from the first program run on subsequent runs, thus accelerating data access and misstating true run times. We corrected this feature by clearing the cache each time we ran the processes.

Time is of critical importance in VM forensics and any method that can reduce the time taken to recover evidence from a VM should be availed of. Our tool indicates that a tailored solution to this issue can have significant benefits in terms of run time reduction.

Conclusion

VM forensics is in its infancy, but with the projected growth in VM use, the problem of forensically examining VMs will only grow too. This problem will become more critical if throw away VMs as described by Barret and Kipper [6] become commonplace. We were careful to ensure that the tool we produced impacted the system being examined in a very insignificant way by writing just one file to RAM disk. We have shown that our tool has a number of important qualities, it executes in an efficient, controlled and verifiable manner, it is simple, fast and forensically sound.

Our approach to VM forensics will allow us to tailor our tool to probe any system, whether it be a VM or standard computer system, any hardware platform or any software platform. It will not be dependent on a compiler or interpreter to be present on the target system, we inject an executable program. We can customise our tool to recover evidence, data, including the password files, log files, PID lists, etc. We are currently examining ways to recover data from open and running processes and ways of cloaking our software execution from a user, presenting a view of the system where it appears only user processes are running.

Our software has a small footprint, it is compact and efficient. One feature of our tool is it's flexibility and we are investigating extending it to OS's other than Linux. As future work we are looking at how best to extract or export our results file from the VM in a forensically secure manner.

References

[1] Casey, E., 2011. Digital evidence and computer crime: Forensic science, computers, and the internet. Academic press

[2] Dykstra, J. and Sherman, A.T., 2012. Acquiring forensic evidence from infrastructure-as-a-service cloud computing: Exploring and evaluating tools, trust, and techniques. Digital Investigation, 9, pp.S90-S98

[3] Goldberg, R.P., 1974. Survey of virtual machine research. Computer, 7(6), pp.34-45

[4] Kremer, J., Cloud Computing and Virtualization. White paper on virtualization

[5] Cusumano, M., 2010. Cloud computing and SaaS as new computing platforms. Communications of the ACM, 53(4), pp.27-29

[6] Barrett, , Kipper,G., Virtualization and Forensics: A Digital Forensic Investigators Guide to Virtual Environments: Syngress, Burlington, Massachusetts, USA, 2010

[7] Cai, H., Wang, N. and Zhou, M.J., 2010, July. A transparent approach of enabling SaaS multi-tenancy in the cloud. In Services (services-1), 2010 6th world congress on (pp. 40-47). IEEE.

[8] http://www.vm.ibm.com/vm40hist.pdf retrieved 25.1.2016

[9] Birk, D., 2011, January. Technical challenges of forensic investigations in cloud computing environments. In workshop on cryptography and security in clouds (pp. 1-6).

[10] Garfinkel, T. and Rosenblum, M., 2003, February. A Virtual Machine Introspection Based Architecture for Intrusion Detection. In NDSS (Vol. 3, pp. 191-206)

[11] Nance, K., Bishop, M. and Hay, B., 2008. Virtual machine introspection: Observation or interference?. IEEE Security & Privacy, (5), pp.32-37

[12] Carrier, B. and Spafford, E.H., 2003. Getting physical with the digital investigation process. International Journal of digital evidence, 2(2), pp.1-20.

[13] Bahram, S., Jiang, X., Wang, Z., Grace, M., Li, J., Srinivasan, D., Rhee, J. and Xu, D., 2010, October. Dksm: Subverting virtual machine introspection for fun and profit. In Reliable Distributed Systems, 2010 29th IEEE Symposium on (pp. 82-91). IEEE

[14] http://www.volatilityfoundation.org/#!about/cmf3, accessed 25.1.2016

[15] Payne, B.D., 2012. Simplifying virtual machine introspection using libvmi. Sandia Report

[16] http://www.cps.gov.uk/legal/assets/uploads/files/ACPO_guidelines_computer_evidence[1].pdf, accessed 27.1.2016

[17] https://www.iso.org/obp/ui/#iso:std:iso-iec:27037:ed-1:v1:en, accessed 27.1.2016

[18] https://www.ncjrs.gov/pdffiles1/nij/199408.pdf, accessed, 27.1.2016

[19] http://ec.europa.eu/anti_fraud/documents/forensics/guidelines_en.pdf, accessed 27.1.2016

[20] https://www.guidancesoftware.com/encase-forensic?cmpid=nav_r

[21] http://digital-forensics.sans.org/

[22] http://accessdata.com/solutions/digital-forensics/forensic-toolkit-ftk

[23] http://www.sleuthkit.org/sleuthkit/

[24] Tobin, P. and Kechadi, T., 2014, March. Virtual machine forensics by means of introspection and kernel code injection. In Proceedings of the 9th International Conference on Cyber Warfare & Security: ICCWS 2014 (p. 294).

[25] Squillante, M.S. and Lazowska, E.D., 1993. Using processor-cache affinity information in shared-memory multiprocessor scheduling. Parallel and Distributed Systems, IEEE Transactions on, 4(2), pp.131-143

[26] http://www.tldp.org/LDP/sag/html/buffer-cache.html, accessed 27.1.2016

[27] Reuther, A., Michaleas, P., Prout, A. and Kepner, J., 2012, September. HPC-VMs: Virtual machines in high performance computing systems. In High Performance Extreme Computing (HPEC), 2012 IEEE Conference on (pp. 1-6). IEEE.

[28] Joshi, A., King, S.T., Dunlap, G.W. and Chen, P.M., 2005, October. Detecting past and present intrusions through vulnerability-specific predicates. In ACM SIGOPS Operating Systems Review (Vol. 39, No. 5, pp. 91-104). ACM

[29] Love, R, 2003. Kernel Korner: CPU affinity. Linux Journal, 2003(111), p.8