FORENSIC ANALYSIS OF VIRTUAL HARD DRIVES

Patrick Tobin
Nhien-An Le-Khac
M-Tahar Kechadi
University College Dublin
Dublin, Ireland
pat.tobin@ucdconnect.ie, {an.lekhac, tahar.kechadi}@ucd.ie

ABSTRACT

The issue of the volatility of virtual machines is perhaps the most pressing concern in any digital investigation. 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. This should be done in the most efficient and secure manner, as quickly, and in a 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 its destruction, is therefore crucial. In this paper, we present a method of 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.

Keywords: Virtual Machine, Digital Forensics, Virtual Machine Forensics, Virtual Hard Drive

INTRODUCTION

It is very rare to find a crime scene where a digital device of some kind 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, and whether they are Unix, Linux or Windows based systems these are devices that can be taken possession of for examination [1]. The data they contain can be catalogued, classified, extracted and subject to detailed examination. They constitute a physical connection between their user or owner, those data they contain 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 an examiner remains the same - to secure as much evidence as possible [2].

How evidence is collected is important to its integrity 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 it 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 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 distributed computing and high speed bandwidth [6]. Our focus is 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 more difficulties.

In this paper, we propose a method of gathering evidence from a VM's vHDD, reduce the data size being gathered, and minimise intrusion on a suspect VM. In the case 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. The 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 with metrics of execution times before and after optimisation. Finally, we will conclude by outlining further research.

2. VIRTUAL MACHINES

VM technologies fall into two categories - Type I and Type II virtual machines, the distinction between these lies in the presence of an underlying OS. Type I virtualisation involves a hypervisor (VMM) using a thin layer of code to allocate 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

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

itself, and provide good performance, availability and security.

Type II hypervisors run as an application on top of an operating system. They are very popular and are usually used to emulate another OS within the OS that the hypervisor is running, e.g. running Windows within Linux, or vice versa. These are more usually found on home computer systems and where security and efficiency is less critical, examples include Oracle VirtualBox, Microsoft VirtualPC and VMware.

2.1 VM Forensics - Current State of 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, has brought with it many challenges for Law Enforcement [9]. VM digital forensics is similar to that of traditional digital forensics, such as log analysis and data capture and analysis, but recovering those data from a cloud VM can pose a challenge. Methods and tools exist to recover data from traditional computer systems and their hard drives, but although the principles are essentially the same, collecting evidence from a vHDD can be more problematic.

In a traditional digital investigation capturing the data on a hard drive involves capturing the suspect computer and seizing and removing the hard drive for analysis, 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 physical 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. It may also expose other users' data on the hard drive, causing privacy concerns, furthermore there are also very few tools to assist in investigating a live vHDD, apart from LibVMI [15]. If the VM is operating on a desktop machine, in VirtualBox or KVM/QEMU, for instance, it may not be possible to gain access to the virtual drive.

2.2 VM Introspection

The most important VM forensics technology developed to date has been Virtual Machine Introspection (VMI) [10]. VMI 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 is now used extensively in the forensic investigation of VMs.

VMI describes how a VMM administrator can inspect that is occurring inside a VM, to view the VM memory, its processes, its network settings, its installed OSes, applications and services. This powerful feature of VMI has allowed criminal investigations of VMs to take place and data to be captured, which would 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 place from outside the guest system being monitored, without the knowledge of that guest system [11]. Furthermore, without knowledge of VMI monitoring it is therefore not possible to prevent it, nor is it 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, this should be avoided as any change to the system being inspected could effectively alter evidence and thus possibly provide a different outcome to that of an unaltered system. This will have consequences for any evidence recovered and may cause that evidence to be ruled as inadmissible. VMI does not affect a VM's performance in any other way, as it does not use any of the VM's resources.

2.2.1 Semantic Awareness

The semantic gap that exists between raw data and its natural language representation, is recognised as the greatest challenge facing virtual machine forensics [30]. Nance et al. [11] describe semantic awareness as the VM's knowledge of its guest operating system (OS), and by Joshi et al. [28] as the level of abstraction used by a virtual machine. Bridging that gap is not a trivial process and is made more difficult by the failure of the OS being inspected to follow certain semantic expectations, it is very much dependent upon the OS following the known data structures and syntax of that OS. By failing to follow those structures and syntaxes 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 and by not adhering to that assumption those data can become subverted. 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 reversal of that obfuscation to be computationally very complex and very expensive, and without prior knowledge of how that 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 in any data being recovered from that OS being rendered unsound, with significant implications for the value of evidence gathered from those data.

2.2.2 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 monitors (VMMs), including VMware and VirtualBox. Linux core dumps can be dumped into ELF files 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 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.

2.2.3 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 is important in the context of VM forensics and our approach to this. These principals have been accepted as best practice by the Courts in the UK, Ireland and Canada, and have influenced the drafting of the EU OLAF guidelines.

3. COLLECTING DATA FROM A VHDD

There are many tools and collection of 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 cloud 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. Also the VM must be live when taking a snapshot rather than the scenario in 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 relevant 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.

When data are recovered from a VM they can be processed in the same manner as those recovered from standard systems. In our proposal, we calculate and recover the md5 signatures of data and propose using these signatures to match against data sets of hash signatures of known files. Matching the recovered hashes against those in repositories such as the National Software Reference Library (NSRL) can identify the files in question where those hash signatures exist in the library. This method of file identification is efficient, because files are identified by means of using a hash signature. In our proposal, we generate MD5's of found files, by doing this we reduce data to be recovered from several MB to 32B. This has advantages in reducing the bandwidth necessary to transmit data, and reducing the volume of data to be stored prior to transmission or recovery. By identifying suspect data through their MD5 hash we can flag those files we need to recover and alert an investigator to their presence. Any alteration to the original data, prior to generating an MD5 hash, will result in a different hash signature to that which would have been generated with original data. This

could be addressed through sub-file forensics, but this is not examined this in this research.

4. EVIDENCE SEARCH THROUGH INJECTED CODE

Our approach to VM forensics involves injecting executable forensic software into a VM and executing that software. In their paper, Tobin and Kechadi [24] described how code injected 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 search engine for this purpose, which will have minimal impact on the host system in terms of processor time consumed, and other resources necessary, e.g. RAM and bandwidth. This engine will simply search a virtual drive, or partition, for pre-defined file types, for example jpegs or documents, and create an MD5 hash of each file found that satisfies the search criteria. The hash signature is then saved 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 interaction with the host system.

Evidence integrity can be compromised by writing to a hard drive. Preventing this happening 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 write-blocker for use with this search engine. We create a small RAM disk, we then install the tool into that RAM disk, execute it from there and save all data found to files within the RAM disk. This prevents any data being written to the vHDD, preserving the vHDD, and because the RAM disk is a reserved area of RAM there are no changes to RAM data. The small size of the RAM disk used, 8 MiB, has very little impact on the VM and its performance.

We believe 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 or save them for extraction by VMI software, and by using the OS semantics this can help bridge the semantic gap. It can help escape kernel data structure manipulation as outlined by Bahram et al [13] by identifying the means of such manipulation, and speed of execution may avoid loss of VM data through shutdown or power-off.

Using the hash signature to help identify files reduces the volume of data for recovery to 32 B per file, from a jpeg of approximately 5 MB, a reduction in data size of approx. 1.5 x 10^4 , giving a very significant reduction in data volume to be extracted. This will result in extraction of a much smaller data footprint, reduce the bandwidth necessary 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, the hash signatures can be used to identify files 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, 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, we should not need to convert data from its raw state to its natural language representation, nor should we have a need to address the data structures. This is a very significant advantage to our approach, as it helps reduce the time needed to examine a system, saves investigator time and reduces the volume of data to be recovered.

We 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 it should be possible to reverse it and recover uncompromised data.

Speed of execution is important whether inspecting a standard computer system or a VM. A standard computer will have a hard drive which can be removed and data recovered from it, but a VM will have no such physical drive, it does not have a persistent physical data store. This is compounded by the volatility of VMs and their storage, delete the VM and everything within the VM is lost. By alerting a user to external activity on their VM this can quickly result in the destruction of the VM and loss of all data. It is therefore important to use code that executes quickly.

5. EXPERIMENT

We describe here what experiments we carried out, and which test data we used.

5.1 Code Optimisation

Optimising code execution is best achieved through careful design of the algorithms, making use of the available hardware, reducing interaction with the user 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 and timing execution.

5.2 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/sec read speed. We used a relatively low power processor to mirror as closely as possible the performance of an Amazon Web Services T2 medium EC2 instance, to measure how our software might operate on such an instance. We gave our VM 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 of this 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.

5.3 Description

To achieve our aims, we built a tool to search the content of a hard drive. The tool searches a file tree for files, recursing into sub-directories when they are found. It then uses the Linux utility *file* to extract the file type, from any files found. We then used the *grep* command to search the output of the file command to identify text files. The program then built the full path to the files found and used the Linux command *md5sum* to calculate the MD5 hash of the files found. The 'md5sum' output is then saved to file.

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 can be tailored to a user's specification by customising the path to be searched and the files to be searched for, however initial testing showed that this approach consumed was CPU heavy, 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 environment. We used this program to make comparisons between its execution time and our program execution time. We have designed our tool to replicate the functions of both find and ftw.h exactly. 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 a termination character is found at which point it will exit the search of that directory branch to resume its search of the parent directory.

5.4 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 one using ftw.h – Tool_2 – ten 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 ten 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. Both tools generated an MD5 hash of files found and saved the resulting hash and the complete file path, to 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, 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 saving. Table 1 illustrates the results we obtained from our tool runs, we have labelled the data appropriately - pinned meaning pinned to one vCPU, unpinned meaning OS managed balancing.

Test Results.

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

	pinned	unpinned
Tool_1	47.48s ± 0.5s	55.59s ± 1.6s
Tool_2	101s ± 2.1s	91s ± 2.5s to10s

Our test results show that our tool ran significantly faster than that using ftw.h and was faster again when CPU affinity was applied. This 'unbalanced' processing environment had an appreciably positive outcome for execution times. An unexpected outcome of our experiments showed that there was smaller divergence from the mean execution time when our tool 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 that our tool was executing as expected. Comparison of the results showed 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 become more obvious.

6. 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 cache memory and reading it from the cache for subsequent operations involving those data, rather than accessing the very much slower memory.

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

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 problem can have significant benefits in terms of run time reduction.

7. CONCLUSION

VM forensics is in its infancy, with the growth in VM use, and its expected future growth, the need to forensically examine VMs will only escalate. We were careful to ensure that the tool we developed 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 quickly. It is simple and forensically sound.

Our approach allows us to tailor our tool to probe any system, whether it is a VM or traditional computer system, any hardware platform or any software platform. It will not be dependent on any compiler, we inject an executable program. We can customise our tool to recover any evidence, any data, including the password files, log files, Process Identifier (PID) lists, etc. We are currently working on ways to recover open and running processes and ways of cloaking our investigative tool 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 its flexibility and we are investigating extending it to interact with OS's other than Linux. As future work, we will build on the strength of the work we present in this paper. We will also be investigating how best

to remove or export the results file from the VM in a forensically secure manner. This is a simple, secure, fast way of recovering data with a reduced risk of corruption of those data. We will also look at the feasibility of extending our approach in memory forensics [31] of mobile devices in smart phone investigations [32].

ABOUT THE AUTHORS

Patrick Tobin is a retired policeman. He has a BSc. in Computer Applications from Dublin City University, Dublin, Ireland and a MSc. in Forensic Computing and Cybercrime Investigation (FCCI) from University College Dublin, Ireland. He is presently (2017) completing his PhD research, his research topic is titled Forensic Evidence Identification and Extraction from Virtual Machines. He is currently lecturing the VoIP module in the MSc. in FCCI.

Dr. Nhien An Le Khac is a Lecturer at the UCD School of Computer Science. He is currently the Program Director of MSc programme in Forensic Computing and Cybercrime Investigation - an international distance learning programme for the law enforcement officers specialising in cybercrime investigations. Previously, 2008, he was a Research Fellow in Citibank, Ireland (Citi). He was also a postdoctoral fellow from 2006 in UCD. He obtained his Ph.D. in Computer Science in 2005 at the Institut National Polytechnique Grenoble (INPG), France. His research interest spans the area of Data Mining/Distributed Data Mining for Security, Fraud and Criminal Detection, Cloud Security and Privacy, Grid and High Performance computing. He has published more than 60 scientific papers in international peer-reviewed journal and conferences in related disciplines. He has also been on the Programme Committee of International Conferences and a regular reviewer for Future Generation of Computer System journal (FGCS, Elsevier).

Professor M-Tahar Kechadi was awarded his PhD and Masters degree - in Computer Science from University of Lille 1, France. He joined the UCD School of Computer Science & Informatics (CSI) in 1999. He is currently Professor of Computer Science at CSI, UCD. His research interests span the areas of Data Mining, distributed data mining heterogeneous distributed systems, Grid and Cloud Computing, and digital forensics and cyber-crime investigations. Prof Kechadi has published over 260 research articles in refereed journals and conferences. He serves on the scientific committees for a number of international conferences and he organised and hosted one of the leading conferences in his area. He is currently an editorial board member of the Journal of Future Generation of Computer Systems and of IST Transactions of Applied Mathematics-Modelling and Simulation. He is a member of the communication of the ACM journal and IEEE computer society.

REFERENCES

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

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

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

[4] Kremer, J. (2010). 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), 27-29.

[6] Barrett, D., & Kipper, G. (2010). Virtualization and forensics: A digital forensic investigator's guide to virtual environments. Syngress.

[7] Cai, H., Wang, N., & 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] Bitner, B., & Greenlee, S. (2012). z/VM A Brief Review of Its 40 Year History.

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

[10] Garfinkel, T., & Rosenblum, M. (2003, February). A Virtual Machine Introspection Based Architecture for Intrusion Detection. In Ndss (Vol. 3, No. 2003, pp. 191-206).

[11] Nance, K., Bishop, M., & Hay, B. (2008). Virtual machine introspection: Observation or interference?. IEEE Security & Privacy, 6(5).

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

[13] Bahram, S., Jiang, X., Wang, Z., Grace, M., Li, J., Srinivasan, D., ... & 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] The Volatility Foundation (2013 - 2014) Retrieved from http://www.volatilityfoundation.org/

[15] Payne, B. D. (2012). Simplifying virtual machine introspection using libvmi. Sandia report, 43-44.

[16] Wilkinson, S., (2012). Good practice guide for computer-based electronic evidence. Association of Chief Police Officers.

[17] Guidelines for identification, collection, acquisition and preservation of digital evidence, (2012), Retrieved from https://www.iso.org/obp/ui/#iso:std:iso-iec:27037:ed-1:v1:en

[18] Ashcroft, J., Daniels, D., Hart, S., (April 2004). NIJ Special Report, (April 2004) Retrieved from https://www.ncjrs.gov/pdffiles1/nij/199408.pdf

[19] Kessler, G., (2016, February, 15th.), Guidelines on Digital Forensic Procedures for OLAF Staff, Retrieved from http://ec.europa.eu/anti_fraud/documents/forensics/guidelines_en.pdf

[20] EnCASE® Forensic (1997 - 2016), Retrieved from forensic?cmpid=nav_r

[21] SANS DFIR (2016) Retrieved from http://digital-forensics.sans.org/

[22] Forensic Toolkit (FTK) (2016), from http://accessdata.com/solutions/digital-forensics/forensic-toolkit-ftk

[23] Carrier, B, (2013 - 2016) The Sleuthkit, Overview, Retrieved from http://www.sleuthkit.org/sleuthkit/

[24] Tobin, P., & Kechadi, T. (2014, January). 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., & Lazowska, E. D. (1993). Using processor-cache affinity information in shared-memory multiprocessor scheduling. Ieee transactions on parallel and distributed systems, 4(2), 131-143.

[26] Wirzenius, Lars, Oja, J., Stafford, S., Weeks, A., (2016, 27th. January), Linux Systems Administrators Guide, Chapter 6 Memory Management, retrieved from http://www.tldp.org/LDP/sag/html/buffer - cache.html, accessed

[27] Reuther, A., Michaleas, P., Prout, A., & 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., & 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), 8.

[30] Dolan-Gavitt, B., Leek, T., Zhivich, M., Giffin, J. and Lee, W., (2011, May). Virtuoso: Narrowing the semantic gap in virtual machine introspection. In Security and Privacy (SP), 2011 IEEE Symposium on (pp. 297-312). IEEE.

[31] Witteman, R., Meijer, A., Kechadi, M. T., & Le-Khac, N. A. (2016, April). Toward a new tool to extract the Evidence from a Memory Card of Mobile phones. In Digital Forensic and Security (ISDFS), 2016 4th International Symposium on (pp. 143-147). IEEE.

[32] Faheem, M., Kechadi, M., & Le-Khac, N. A. (2016). The State of the Art Forensic Techniques in Mobile Cloud Environment: A Survey, Challenges and Current Trends. arXiv preprint arXiv:1611.09566.

