

**DIGITAL FORENSIC ACQUISITION OF VIRTUAL PRIVATE
SERVERS HOSTED IN CLOUD PROVIDERS THAT USE KVM AS
A HYPERVISOR**

by

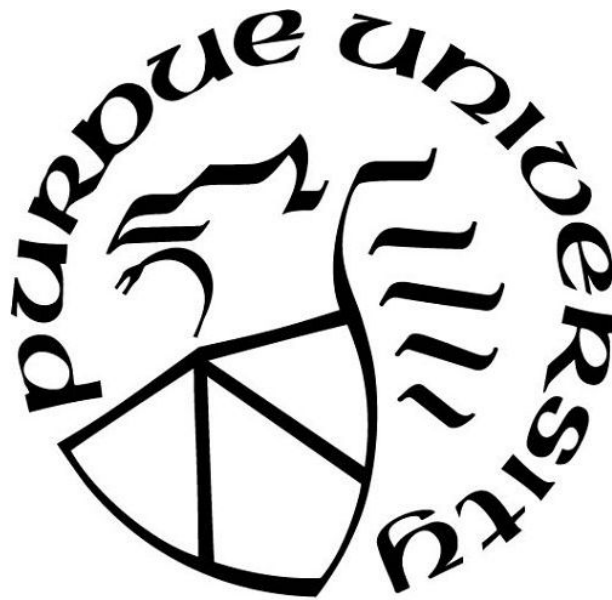
Adolfo A. Montironi

A Thesis

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the Degree of

Master of Science



Department of Computer and Information Technology

West Lafayette, Indiana

August 2018

ProQuest Number: 10845501

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



ProQuest 10845501

Published by ProQuest LLC (2018). Copyright of the Dissertation is held by the Author.

All rights reserved.

This work is protected against unauthorized copying under Title 17, United States Code
Microform Edition © ProQuest LLC.

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 – 1346

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Marcus K. Rogers, Chair

Department of Computer and Information Technology

Dr. Kathryn C. Seigfried-Spellar

Department of Computer and Information Technology

Dr. John A. Springer

Department of Computer and Information Technology

Approved by:

Dr. Eric T. Matson

Head of the Graduate Program

Dedicated to my lovely wife, Patricia, for her unconditional support and encouragement during this journey, and to my two children, Sofía and Leonardo, who inspire me to become a better person every day.

ACKNOWLEDGMENTS

I wish to gratefully acknowledge all the faculty members of Purdue University for their support and guidance throughout the master's program. I would like to especially thank Dr. Marcus K. Rogers for the numerous meetings, his insightful comments, and inestimable help during this research. I would also like to thank Dr. Kathryn C. Seigfried-Spellar and Dr. John A. Springer for agreeing to be part of my committee and for helping and supporting me throughout the process.

I would also like to acknowledge my home-country, Argentina, for giving me the opportunity of studying abroad during two years in order to complete this master's program, and to the United States of America, a great and beautiful country that made me feel welcome since the day I arrived.

Last but not least, I would like to thank my family and friends, specially my mother, Gladys, and my grandfather, Oscar, for their love, support, and encouragement since the first day.

TABLE OF CONTENTS

LIST OF TABLES	ix
LIST OF FIGURES	x
LIST OF ABBREVIATIONS	xi
GLOSSARY	xiii
ABSTRACT	xiv
CHAPTER 1. INTRODUCTION	1
1.1 Background	1
1.2 Scope	4
1.3 Significance	5
1.4 Research Question and Hypotheses	7
1.5 Assumptions	7
1.6 Limitations	8
1.7 Delimitations	9
1.8 Summary	10
CHAPTER 2. REVIEW OF RELEVANT LITERATURE	11
2.1 Cloud Computing	11
2.1.1 Definition and Characteristics	11
2.1.2 Service Models and Deployment Models	12
2.2 Virtualization	13
2.2.1 Definition and Concepts	13
2.2.2 KVM	14
2.2.2.1 Virtual storage	16
2.2.2.2 Virtual storage snapshots and images	19
2.2.2.3 Virtual RAM	20
2.2.2.4 Virtual RAM images	21
2.2.2.5 Virtual networking	22
2.2.2.6 Virtual network traffic capturing	24
2.2.3 VPS	26

2.3	Cloud Computing and Virtualization in Digital Forensics	27
2.3.1	Early Research	27
2.3.2	Cloud Computing and Virtualization	28
2.3.3	Complementary Approaches	32
2.3.4	Virtual Machine Introspection	33
2.4	Summary	35
2.5	This Research Study	36
CHAPTER 3. METHODOLOGY		37
3.1	Research Question and Hypotheses	37
3.2	Participants	38
3.3	Research Design	38
3.3.1	Hardware Specifications	40
3.3.1.1	Virtualization node	40
3.3.1.2	Laptop	41
3.3.1.3	Switch and Ethernet patch cords	41
3.3.2	Virtualization Node Implementation	41
3.3.2.1	Operating system installation	41
3.3.2.2	KVM installation and configuration	44
3.3.2.3	Additional configuration steps	44
3.3.3	Virtual Machine Deployment	47
3.3.3.1	VM1 - Ubuntu Server 17.10 (32-bit)	48
3.3.3.2	VM2 - CentOS 7 (64-bit)	49
3.3.3.3	VM3 - Windows Server 2008 Standard (32-bit)	51
3.3.3.4	VM4 - Windows Server 2016 Standard (64-bit)	52
3.3.3.5	Additional configuration steps	53
3.3.4	Laptop Configuration	54
3.4	Testing Conditions	55
3.4.1	Hard drives transfer rates	56
3.4.1.1	Read transfer rate for /dev/sda	56
3.4.1.2	Write transfer rate for /dev/sda	58

3.4.1.3	Read transfer rate for /dev/sdb	60
3.4.1.4	Write transfer rate for /dev/sdb	61
3.4.1.5	Data duplication transfer rate from /dev/sda to /dev/sdb	62
3.4.2	RAM data duplication transfer rate	64
3.5	Testing Procedures	66
3.5.1	Virtual hard drive image creation	66
3.5.2	RAM image creation and network traffic capturing	68
3.6	Measurements for Evaluation	71
3.6.1	Hypothesis One	71
3.6.2	Hypothesis Two	74
3.6.3	Hypothesis Three	75
3.7	Measurements for Success	77
3.7.1	Hypothesis One	77
3.7.2	Hypothesis Two	77
3.7.3	Hypothesis Three	78
3.8	Threats to Validity	79
3.9	Summary	80
CHAPTER 4.	RESULTS	81
4.1	Hypothesis One	81
4.2	Hypothesis Two	86
4.3	Hypothesis Three	91
4.4	Summary	94
CHAPTER 5.	DISCUSSION	95
5.1	Procedure to acquire digital evidence from a VPS hosted in KVM	96
5.2	Significance	100
5.3	Limitations	102
5.4	Recommendations for Future Studies	102
5.5	Summary	104
REFERENCES	105
APPENDIX A.	PACKAGES INSTALLED ON THE VIRTUALIZATION NODE .	110

APPENDIX B. CONFIGURATION FILES OF THE VIRTUALIZATION NODE .	123
APPENDIX C. VIRTUAL MACHINES CONFIGURATION FILES	125
APPENDIX D. GENERATION OF VOLATILITY PROFILES	138
APPENDIX E. BASH SCRIPT TO TEST THE IMAGING PROCESS	140
APPENDIX F. PURDUE’S IRB EXEMPTION LETTER	143
APPENDIX G. USER INTERACTION SCRIPT FOR VM1 (UBUNTU 17.10) . .	144
APPENDIX H. USER INTERACTION SCRIPT FOR VM2 (CENTOS7)	147
APPENDIX I. USER INTERACTION SCRIPT FOR VM3 (WINDOWS 2008) . .	151
APPENDIX J. USER INTERACTION SCRIPT FOR VM4 (WINDOWS 2016) . .	154

LIST OF TABLES

2.1	Storage pool, allocation policy, and virtual hard drive format supported by KVM	18
3.1	Partition table of device /dev/sda	43
3.2	Virtual machine specifications	47
3.3	ISO files used to install the OS in each VM	48
3.4	Read transfer rate for device /dev/sda	57
3.5	Write transfer rate for device /dev/sda	59
3.6	Read transfer rate for device /dev/sdb	60
3.7	Write transfer rate for device /dev/sdb	61
3.8	Data duplication transfer rate from /dev/sda to /dev/sdb	63
3.9	RAM data duplication transfer rate	65
3.10	Steps followed by the participants to interact with the VMs	69
3.11	Steps and expected results of the virtual hard drive image creation process .	72
3.12	Steps and expected results of the RAM image creation process	74
3.13	Steps and expected results of the network traffic capturing process	76
4.1	VM1 hard drive image creation process	82
4.2	VM2 hard drive image creation process	83
4.3	VM3 hard drive image creation process	84
4.4	VM4 hard drive image creation process	85
4.5	VM1 RAM image creation process	87
4.6	VM2 RAM image creation process	88
4.7	VM3 RAM image creation process	89
4.8	VM4 RAM image creation process	90
4.9	VM1 network traffic capturing process	92
4.10	VM2 network traffic capturing process	92
4.11	VM3 network traffic capturing process	93
4.12	VM4 network traffic capturing process	93

LIST OF FIGURES

2.1	KVM virtualization node components	15
2.2	KVM virtual network diagram	23
2.3	KVM virtualization node and VPSs overview	26
3.1	Research environment	39
3.2	Partition layout for /dev/sda and /dev/sdb	45
3.3	/dev/sda read transfer rate	58
3.4	/dev/sda write transfer rate	59
3.5	/dev/sdb read transfer rate	61
3.6	/dev/sdb write transfer rate	62
3.7	Data duplication transfer rate from /dev/sda to /dev/sdb	63
3.8	RAM data duplication transfer rate	66
3.9	Network connections established from the laptop by the participants	69
F.1	Purdue’s Institutional Review Board exemption letter	143
G.1	Ping command screen	144
G.2	Putty screen	145
G.3	Chrome screen	146
H.1	Ping command screen	147
H.2	Putty screen	148
H.3	FileZilla Site Manager screen	149
H.4	FileZilla screen once the FTP connection is established	150
I.1	Ping command screen	151
I.2	Remote desktop client screen	152
I.3	Nslookup command screen	153
J.1	Ping command screen	154
J.2	Remote desktop client screen	155
J.3	MySQL Workbench screen	156
J.4	MySQL Workbench screen once the MySQL connection is established	156

LIST OF ABBREVIATIONS

b	bit
B	byte (8 bits)
BASH	Bourne-again shell
CaaS	crime as a service
CPU	central processing unit
DDoS	distributed denial of service
DDR	double data rate type two
FB-DIMM	fully-buffered dual in line memory module
GB	gigabyte (1,000 MB)
Gbit	gigabit (1,000 Mb)
GHz	gigahertz (1,000 MHz)
GiB	gibibyte (1,024 MiB)
Hz	hertz
IaaS	infrastructure as a service
iSCSI	Internet small computer system interface
KVM	kernel-based virtual machine
LVM	logical volume manager
Mb	megabit (1,000 bits)
MB	megabyte (1,000 bytes)
MHz	megahertz (1,000 Hz)
MiB	mebibyte (1,024 bytes)
NFS	network filesystem
OS	operating system
PaaS	platform as a service
RAM	random access memory
RPM	revolutions per minute
SaaS	software as a service
SATA	serial advanced technology attachment

sec	second
TB	terabyte (1,000 GB)
TiB	tebibyte (1,000 GiB)
USB	universal serial bus
VM	virtual machine
VMs	virtual machines
VMI	virtual machine introspection
VPS	virtual private server
VPSs	virtual private servers

GLOSSARY

Bit-stream image – an exact bit by bit copy created from a storage device such as a hard drive.

Cloud computing – “a model for enabling ubiquitous convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction” (Mell & Grance, 2011)

Cloud computing provider – a company that owns and maintains the cloud computing resources rented by customers (Garcia, 2014)

Virtualization – “it is the process of hiding the underlying physical hardware so that it can be shared and used by multiple virtual machines” (Chirammal, Mukhedkar, & Vettathu, 2016) ; It is an abstraction layer that allows the creation of software representations of computer physical resources (Pollitt et al., 2008)

Virtualization node – “the physical computer that runs the virtualization software” (Chirammal et al., 2016)

Hypervisor – “a layer between the underlying physical hardware and the virtual machines running on top of it [...] It is a piece of software that is responsible for monitoring and controlling virtual machines” (Chirammal et al., 2016)

Virtual machine (VM) – is a software emulation of a physical computer that runs its own operating system and applications (Pollitt et al., 2008)

Kernel-based virtual machine (KVM) – is a full virtualization solution that turns the Linux kernel into a hypervisor by inserting a loadable kernel module (Chirammal et al., 2016)

Virtual machine introspection (VMI) – a technique that “consists in monitoring virtual machines from the hypervisor level in order to inspect their states and activities” (Hebbal, Laniepce, & Menaud, 2015)

Virtual private server (VPS) – a cloud computing solution that consists of a private virtual machine, managed exclusively by a customer (Zahedi, 2014)

ABSTRACT

Author: Montironi, Adolfo A.. M.S.

Institution: Purdue University

Degree Received: August 2018

Title: Digital Forensic Acquisition of Virtual Private Servers Hosted in Cloud Providers
that Use KVM as a Hypervisor

Major Professor: Marcus K. Rogers

Kernel-based Virtual Machine (KVM) is one of the most popular hypervisors used by cloud providers to offer virtual private servers (VPSs) to their customers. A VPS is just a virtual machine (VM) hired and controlled by a customer but hosted in the cloud provider infrastructure. In spite of the fact that the usage of VPS is popular and will continue to grow in the future, it is rare to find technical publications in the digital forensic field related to the acquisition process of a VPS involved in a crime. For this research, four VMs were created in a KVM virtualization node, simulating four independent VPSs and running different operating systems and applications. The utilities `virsh` and `tcpdump` were used at the hypervisor level to collect digital data from the four VPSs. The utility `virsh` was employed to take snapshots of the hard drives and to create images of the RAM content while the utility `tcpdump` was employed to capture in real-time the network traffic. The results generated by these utilities were evaluated in terms of efficiency, integrity, and completeness. The analysis of these results suggested both utilities were capable of collecting digital data from the VPSs in an efficient manner, respecting the integrity and completeness of the data acquired. Therefore, these tools can be used to acquire forensically-sound digital evidence from a VPS hosted in a cloud provider's virtualization node that uses KVM as a hypervisor.

CHAPTER 1. INTRODUCTION

This chapter presents the introduction to this research study. The general background, scope, and significance of the problem that lead to the research question and hypotheses are described next. Limitations, delimitations, and assumptions are also introduced.

1.1 Background

Digital forensics is a science whose main objective is to gather and examine evidence from digital devices in order to be presented in a court of law during a criminal investigation (Holt, Bossler, & Seigfried-Spellar, 2015). It is relatively young compared to other forensic sciences such as DNA or entomology, but its contribution is vital because nowadays every crime involves some kind of digital information (Holt et al., 2015). A digital device can be involved in a crime in three different ways: as a tool to commit the crime, as a target of the criminal behavior, and as an incidental to the illegal activity (Holt et al., 2015; Rogers, 2017). Although there is no unique standard procedure to depict the phases of a digital investigation process in every case, there are four common phases always present: identification, acquisition, examination, and presentation (Holt et al., 2015). During identification, investigators survey the digital environment to detect possible sources of data (Holt et al., 2015). Acquisition is the process of collecting data from the sources previously identified, while examination represents the analysis of the data to extract meaningful evidence (Holt et al., 2015). Finally, during the presentation phase investigators report the methods used and the results obtained in a court of law (Holt et al., 2015).

The acquisition phase is crucial because the success of the following phases depends on the data collected (Kolhe & Ahirao, 2017; Ligh, Case, Levy, & Walters, 2014). The investigator has to decide which data need to be collected and the proper method to do it in order to preserve the state of the digital environment under examination (Kolhe & Ahirao, 2017; Ligh et al., 2014). This means the acquisition has to be

completed in a forensically-sound manner to ensure the evidence does not lose its admissibility and evidentiary value when presented in a court of law (Lessing & von Solms, 2008; McKemmish, 2008). To this end, the acquisition process has to be reliable and accurate in order to guarantee two fundamental properties of the data acquired: integrity and completeness (Beebe, 2009; Holt et al., 2015; Lessing & von Solms, 2008; McKemmish, 2008). Integrity is achieved by collecting an authentic copy of the data from the digital environment, while at the same time preserving its original state in order to minimize the probability of contamination (Holt et al., 2015). Completeness signifies collecting all the available data to obtain a comprehensive understanding of the state of the digital environment during a particular moment (Holt et al., 2015).

In investigations where traditional computers are involved, these two properties can be achieved by powering the computer off, using a write blocker to create bit-stream images of the hard drives, and verifying that the hash values from the hard drives and the images created match (Holt et al., 2015). This has been considered the traditional procedure to create authentic and complete images from hard drives and to preserve the state of the evidence stored inside them (Lessing & von Solms, 2008; Ligh et al., 2014). However, this procedure is not always the best option. In a time-sensitive investigation such as a kidnapping or a child abduction the traditional procedure has to be modified in order to retrieve critical information in a short period of time, directly at the crime scene (Rogers, Goldman, Mislán, Wedge, & Debroya, 2006). Information related to emails, browser, and instant messaging artifacts should be examined first, instead of creating a bit-stream image of every storage device (Rogers et al., 2006). Furthermore, the traditional forensic procedure has an important disadvantage: volatile data such as RAM content or network traffic are lost when the device is powered off, even though this data represent a fundamental part of the digital environment being analyzed (Lessing & von Solms, 2008; Ligh et al., 2014).

As the digital forensic field has evolved, the need to take into account not only hard drives but also other sources of data (e.g., RAM and network traffic) has become vital to achieve a deeper comprehension of the digital environment (Ligh et al., 2014). Those other sources were omitted in the past because the environment has to be altered in order

to collect data from them and because it is not possible to determine the integrity and completeness of the data acquired in the same way it is for hard drives. Nonetheless, every acquisition method introduces modifications in the digital environment; even the traditional procedure of powering the computer off to create bit-stream images from its hard drives (Kolhe & Ahirao, 2017; Lessing & von Solms, 2008; Ligh et al., 2014). The key to still achieve the integrity and completeness required depends on the investigator's understating on how each method impacts on the environment and how to minimize this impact during the investigation process (Ligh et al., 2014). Detailed documentation of the process followed and the results obtained by the investigator also plays a fundamental role to achieve forensic soundness (Lessing & von Solms, 2008).

In a digital environment, data are available in three different states: at rest, in motion, or in execution (Birk, 2011; Birk & Wegener, 2011). Data at rest are kept in storage devices, data in motion are transferred over a network from one device to another, and data in execution are loaded in the device's RAM in order to be read, modified, or executed (Birk, 2011; Birk & Wegener, 2011). These three states are intimately related to the sources from where data can be collected: hard drives (or other permanent storage devices), RAM, and network interfaces. The procedure to collect information focused only on hard drives and the creation of bit-stream images is referred to as dead forensic acquisition since it is performed while the device is off (Kolhe & Ahirao, 2017). On the contrary, live forensic acquisition includes procedures to retrieve non-persistent (volatile) data from RAM and network interfaces before powering the device off (Kolhe & Ahirao, 2017). The course of action during an acquisition phase should be based on the persistence of the data to be collected, prioritizing the most volatile first (Ligh et al., 2014). For this reason, network traffic should be collected first since it is exchanged in real-time between the devices. Then, the RAM content because it is lost once the device is powered off. Finally, hard drives and other persistent storages which are more stable sources of information.

When mobile devices such as cell phones, tablets, and watches are involved in a investigation, most of the time it is not possible to create bit-stream images from their permanent storages (Owen & Thomas, 2011). Even if it is possible, the devices usually

have to be modified in order to allow a bit-stream extraction, which means the digital environment has been altered. Other limitations apply to cloud computing solutions. The cloud provider's infrastructure is usually shared among different users, and collecting data from a server could affect the privacy of other users not related to the investigation (Garcia, 2014). Even if the privacy issues are ignored and all the data from a shared server is collected, the volume of the data retrieved can be a technical limitation during the examination phase.

With the development of new technologies and the tendency to move solutions to the cloud, the traditional forensic procedures defined for personal computers do not seem to be appropriate enough to justify the digital evidence admissibility. Under this scenario, it seems more appropriate to evaluate the admissibility of the evidence based on the methods used by the investigators and their knowledge and skills to justify their actions. For this reason, the present study evaluated possible methods to collect digital data from a virtual private server (VPS) hosted in a cloud provider virtualization node that uses Kernel-based Virtual Machine (KVM) as a hypervisor.

1.2 Scope

The cloud computing paradigm enables scalable, on-demand, and affordable computing resources through the Internet (Garcia, 2014). Virtualization is a key concept in cloud computing (Garcia, 2014). Cloud providers make use of virtualization by offering VPSs to their customers. A VPS is a private VM managed by the customer but hosted in the cloud provider infrastructure. KVM is one hypervisor used by cloud providers to deliver and host VPSs. Customers pay according to the resource usage and they have total control over their VPS (i.e., they can modify the operating system or install and execute any particular software).

The present study focused on the acquisition process, using two utilities to collect digital data from a VPS hosted in a full virtualized environment that uses KVM as a hypervisor: `virsh` and `tcpdump`. These utilities were selected with a twofold objective. First, they allow the collection of data from the hypervisor level and without needing

access to the VPS itself, which implies a more reliable result. Second, they are common utilities available in the majority of the virtualization nodes based on GNU/Linux distributions and their installation and execution do not require significant modifications to the production environment. Three different sources of digital data from a VPS were taken into account to analyze the performance of the utilities: hard drives, RAM, and network interfaces. These three sources were selected because they represent the most fruitful locations of potential evidence from a digital forensic perspective.

The performance of the utilities was analyzed in terms of efficiency, integrity, and completeness because these three characteristics are critical to ensure the acquisition process is accurate and reliable. Efficiency is measured according to the time needed to collect the data. Integrity is achieved when the data collected are an authentic copy of the original source and the source has not been modified during the process (Holt et al., 2015). Completeness is accomplished when all the data from a specific source of evidence are acquired (Holt et al., 2015). Although this study focused exclusively on VPSs, the results and conclusions of using these utilities for acquisition purposes are valid also for any VM hosted in a KVM virtualization node.

1.3 Significance

The global cloud computing market grew 21% to achieve \$110 billion in 2015 (SynergyResearchGroup, 2016). In 2016, spending on hiring public cloud computing infrastructure reached \$38 billion and it is forecast to grow to \$173 billion in 2026 (Forbes, 2016). The virtualization market was valued at \$10 billion in 2014 and it is expected to reach \$21.5 billion by 2019, as the virtualization market increases in response to demands for cloud services (Technavio, 2015). In 2016, 17% of enterprises run over 1,000 VPSs in the public cloud, compared to 13% in 2015 (RightScale, 2016). The previous indicators suggest the usage of cloud computing and virtualization solutions will continue to grow during the next years.

Even though VMware solutions lead the general hypervisor market share, when the focus is on cloud computing providers the results are different. Cloud providers cannot

afford the costs associated to VMware if they want to offer a lower price to their customers. Instead, they rely on open source virtualization projects such as Xen or KVM because they offer an excellent performance and they do not have license costs associated. Digital Ocean, the third largest cloud provider, uses KVM as a hypervisor (Chirammal et al., 2016). Amazon Web Services (AWS), the largest IaaS cloud provider, has been using Xen as principal hypervisor for several years but recently it announced a shift to KVM for future EC2 VPSs (TheRegister, 2017). These indicators imply the usage of KVM will continue to grow in the near future and become the most preferred hypervisor by cloud providers.

For these reasons, it is of paramount importance to define digital forensic procedures to technically deal with VPSs hosted in a KVM virtualization node. Different studies have been conducted to examine technical and legal general implications of cloud computing, but there is no research focused particularly on KVM and the details of the acquisition process of VPSs.

Furthermore, the cloud computing characteristics of global access, faster provisioning, high control, and affordability pave the way for criminals to access computing resources and perform illegal activities. They can rent their own VPSs using stolen credit card information or they can exploit vulnerable accounts to be used as launch points to commit offenses. In fact, the term Crime as a Service (CaaS) has been introduced to define this type of illegal cloud-based services offered by criminals, such as on-demand distributed denial of service (DDoS) attacks (BankInfoSecurity, 2017). The researcher has worked with different cloud providers and is well aware of how a this problematic negatively affects not only their business, but also the victims of these illegal activities.

It is expected that the results of this research will guide forensic practitioners and cloud providers to collect forensically-sound digital evidence from a VPS involved in a crime and hosted in a virtualization node that uses KVM as a hypervisor. Moreover, it is the intent of this study to contribute to enlarge the body of knowledge of digital forensics in this particular area and to reduce the criminal cases that involve the usage of VPSs.

Similar research could be conducted on other hypervisors such as XEN, OpenVZ, or Hyper-V. Forensic software companies could use these results to develop remote agents

to be executed in cloud provider's virtualization nodes in order to collect data from VPSs. If these agents were developed as open source projects, the details of their functioning would be known, which could potentially increase the willingness of the cloud provider to cooperate with the investigation. The agents could collect data locally or transfer it remotely using encryption mechanisms. The data collected from the VPSs could be integrated into the existent forensic software suites to provide a unified examination interface to investigators. In addition, once the practical details of the acquisition phase in distinct hypervisors are extensively analyzed, the global legislation on this topic could be updated in order to deal with this problem also from a legal standpoint.

1.4 Research Question and Hypotheses

The principal objective of this research was to answer the following research question: is it possible to acquire forensically-sound digital evidence from a VPS hosted in a cloud provider's virtualization node that uses KVM as a hypervisor?

The hypotheses of this study were stated as follows:

H₁: images of the hard drives of a VPS can be created in an efficient manner, respecting the integrity and completeness of the data acquired.

H₂: images of the RAM content of a VPS can be created in an efficient manner, respecting the integrity and completeness of the data acquired.

H₃: the network traffic of a VPS can be captured in real-time, respecting the integrity and completeness of the data acquired.

1.5 Assumptions

The assumptions of this study were:

1. The research environment simulated a VPS provider's infrastructure, using a physical computer as a virtualization node.

2. The virtualization node hosted four VMs that simulated four independent VPSs.
3. A laptop simulated a system used by different participants to interact with the four VPSs.
4. The virtualization node and the laptop were connected to an Ethernet switch in order to allow network communication between them and with the four VPSs.
5. Administrator access to the virtualization node was granted in order to complete the acquisition process of each VPS.
6. The processes of installing and executing the utilities `virsh` and `tcpdump` did not require significant modifications to the virtualization node and to the VPS provider's environment.
7. The hardware components and software used throughout the study worked appropriately.
8. The methodology and results of this research could be applied to any virtualization node as long as it is configured in the same manner and using the same software versions.

1.6 Limitations

The limitations of this study were:

1. The study was conducted using a GNU/Linux Ubuntu Server 16.04.4 64-bit system (kernel version: 4.4.0-116.140) to simulate a virtualization node of a VPS provider.
2. The hypervisor used throughout the study was KVM.
3. The KVM module loaded into the kernel was provided by the `linux-image-generic` package (version 4.4.0-116.140).
4. The KVM user-space tools were provided by the `qemu-kvm` package (version 2.5.0).

5. The libvirt service was provided by the libvirt0 package (version 1.3.1-1). The libvirt utilities, including virsh, were provided by the libvirt-bin package (version 1.3.1-1).
6. The research environment included four VPS running different operating systems: Ubuntu Server 17.10 (32-bit), CentOS 7 (64-bit), Windows Server 2008 Standard (32-bit), and Windows Server 2016 Standard (64-bit).
7. The study tested utilities to collect digital data from three possible sources of each VPS: hard drives, RAM, and network interfaces.
8. The utility virsh (version 1.3.1-1) was used to create images from the virtual hard drives and from the RAM content of each VPS.
9. The utility tcpdump (version 4.9.2-0) was used to capture in real-time the network traffic of each VPS.
10. The data collection was performed under low load conditions on the virtualization node.

1.7 Delimitations

The delimitations of this study were:

1. Other operating systems were not considered to simulate a virtualization node of a VPS provider.
2. Other hypervisors, KVM (module or user-space tools) versions, and libvirt (service or client) versions were not considered.
3. The utility virsh (version 1.3.1-1) was the only utility used to create images from the virtual hard drives and from the RAM content of each VPS.
4. The utility tcpdump was the only utility used to capture in real-time the network traffic of each VPS.

5. Instead of using a public IP address, each VPS was assigned with a private IP address.
6. The data collection was not performed under heavy load conditions on the virtualization node.

1.8 Summary

This chapter provided the general background, scope, significance, research question and hypotheses, assumptions, limitations, and delimitations for this research study. The following chapter provides a review of the literature relevant to cloud computing and virtualization from a digital forensic perspective.

CHAPTER 2. REVIEW OF RELEVANT LITERATURE

This chapter aims to review existent digital forensic research on cloud computing and virtual environments. The first section defines the principal ideas of the cloud computing paradigm. The second section provides the fundamental concepts of virtualization and explores the KVM internals for the purposes of the present study. The third section analyzes the latest and most significant research on cloud computing and virtualization. The fourth section summarizes the significant conclusions drawn from the research and the fifth section introduces the approach taken by this research study.

2.1 Cloud Computing

2.1.1 Definition and Characteristics

It seems appropriate to start this review with the principal concepts related to cloud computing. The most generally accepted definition of cloud computing was provided by Mell and Grance (2011):

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. (p. 2)

From this definition, it can be stated that cloud computing has five inherent characteristics: on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service (Mell & Grance, 2011). On-demand self service means that a customer can hire computing resources from the cloud provider without human interaction. Broad network access implies that the services can be accessed over the Internet using different devices such as cell phones, tablets, or laptops. Resource pooling

means the provider offers computer resources dynamically to attend the needs of each customer. Rapid elasticity implies the resources are delivered and released simply according to the consumer demand. Measured services means the resources are monitored, administered, and billed based on utilization.

2.1.2 Service Models and Deployment Models

Cloud computing can be classified into three different service models:

Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) (Mell & Grance, 2011). IaaS provides processing, storage, network, and other fundamental computing resources to the customers where they can install and execute all their necessary software, including the operating system and applications. The customer does not manage the infrastructure of the cloud but has total control over the operating system and applications deployed, and over the usage of processor, storage, and network resources (Mell & Grance, 2011). Amazon Web Services (AWS) and Rackspace are two examples of IaaS providers. PaaS provides the cloud infrastructure where customers can deploy their own or acquired applications. The customer does not manage the infrastructure, the operating system, or the usage of processor, storage, and network resources, but has full control over the applications deployed (Mell & Grance, 2011). Google App Engine and Microsoft Azure are two examples of PaaS providers. SaaS provides applications deployed by the provider onto its own infrastructure to be used by the customers. The customer does not manage the infrastructure of the cloud, the operating system, or the applications deployed, with the exception of user-side configuration settings (Mell & Grance, 2011). Microsoft Office 365 is an example of SaaS.

Cloud computing can also be classified into four different deployment models: private, community, public, and hybrid (Mell & Grance, 2011; Reilly, Wren, & Berry, 2010). Private cloud is used exclusively by a particular organization which owns, manages, and operates the cloud infrastructure. Community cloud is controlled exclusively by a specific community whose members share common concerns. Public

cloud is open used by the general public such as business, academic, and government organizations, or a combination of them. Hybrid cloud is a combination of two or more different models of cloud previously described.

The present study focused particularly on the IaaS model because it uses virtualization as the key component, it permits the highest level of customization by the customers, and it also represents the most fruitful scenario to find potential evidence. The IaaS model is also the preferred option by most cybercriminals because they can rent this service using stolen credit card information, or they can exploit vulnerable accounts to be used as launch points for their illegal activities. A clear example of this predicament was the attack on Sony's PlayStation servers, which was launched from an Amazon's Elastic Compute Cloud (Amazon EC2) server, and affected over 100 million customers (ITProPortal, 2011). Furthermore, the term Crime as a Service (CaaS) has been introduced to define this type of illegal services offered by criminals such as on-demand distributed denial of service (DDoS) attacks (BankInfoSecurity, 2017).

2.2 Virtualization

2.2.1 Definition and Concepts

Virtualization is the key component of the IaaS model (Garcia, 2014). It is an abstraction layer that allows the creation of multiple virtual (software-based) resources from physical computer resources (Pollitt et al., 2008). Once created, these virtual resources can be allocated to a virtual machine (VM), which is an isolated process being executed by the virtualization software on the physical computer (Pollitt et al., 2008). The VM behaves like an independent computer, using the assigned virtual resources and running its own operating system (OS) and applications (Bem & Huebner, 2007). The physical computer from where the virtual resources are created is called virtualization node or host, while the VMs are called guests or domains (Barrett & Kipper, 2010; Pollitt et al., 2008). The virtualization software is called hypervisor (also referred to as

virtual machine monitor or VMM) and it is responsible for controlling the interaction between the VMs and the physical host (Barrett & Kipper, 2010; Pollitt et al., 2008).

There are two principal modes of virtualization: paravirtualization and full virtualization. In paravirtualization, the VM OS is aware it runs in a virtual environment, so it modifies some internal operations to communicate with the hypervisor instead of establishing a direct communication with the different devices (Pollitt et al., 2008; RedHat, 2017). In full virtualization, the VM OS is not aware it runs in a virtual environment because the devices are virtualized by the hypervisor and the VM OS can establish a direct communication with them (Pollitt et al., 2008; RedHat, 2017). Hypervisors that run on top of the host in place of an OS, such as VMware ESX, are called Type I hypervisors (Barrett & Kipper, 2010; Pollitt et al., 2008). On the contrary, hypervisors that run as a process inside the host OS, such as Kernel-based Virtual Machine (KVM), Xen, and Hyper-V, are called Type II hypervisors (Chirammal et al., 2016; Pollitt et al., 2008).

2.2.2 KVM

KVM is an open source, hardware-assisted, full virtualization solution. It was designed to be a modern hypervisor that takes advantage of CPU virtualization technologies such as Intel VT-x and AMD-V (Chirammal et al., 2016). It can be installed in most GNU/Linux distributions and it works by loading a module into the Linux kernel to convert the system into a hypervisor (Chirammal et al., 2016). The complete KVM virtualization solution includes the following components:

- KVM module: it is a Linux loadable kernel module, which provides hardware-assisted full virtualization support making use of the virtualization extensions offered by the underlying CPU (Chirammal et al., 2016).
- Quick Emulator (QEMU): it is a user-space software that provides emulation of CPUs and peripheral devices such as hard drives, network interfaces, video cards, and USB ports (Chirammal et al., 2016). QEMU utilizes the KVM kernel module

to achieve real hardware-assisted full virtualization support instead of only emulation (Chirammal et al., 2016). The properties of each VM such as number of CPUs, RAM size, and virtual devices assigned are specified in a XML configuration file located inside the directory `/etc/libvirt/qemu` (Chirammal et al., 2016).

- **Libvirt:** it is a management layer that runs as a service and communicates with different hypervisors, including KVM (Chirammal et al., 2016). Libvirt provides an application programming interface (API) to manage the VMs and the functionalities of the hypervisor (RedHat, 2017). It spawns one QEMU process per each VM, with the specifications defined in its corresponding XML configuration file (Chirammal et al., 2016).
- **Libvirt client:** it is the application used to connect to the Libvirt service (Chirammal et al., 2016). Virsh is the principal Libvirt client and it is an advanced command-line utility that allows to manage the VMs and control the functioning of the hypervisor (RedHat, 2017).

Figure 2.1 illustrates all the components previously described and their relationship in a GNU/Linux virtualization node that uses KVM as a hypervisor.

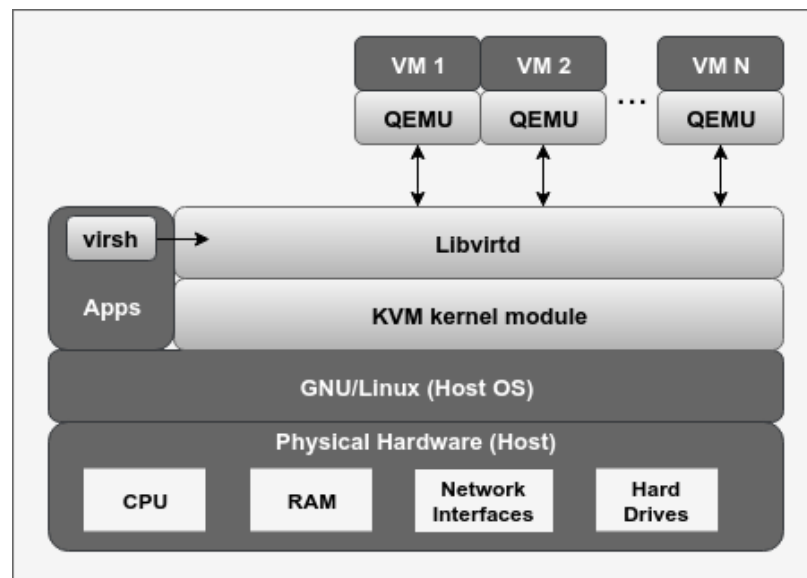


Figure 2.1. KVM virtualization node components

2.2.2.1 Virtual storage

The first concept to define related to storage management in KVM is storage pool. A storage pool represents reserved storage space on the virtualization node, that can be used to create virtual hard drives and assign them to VMs (Libvirt, 2018). By default, only one storage pool is defined and it is the directory `/var/lib/libvirt/images`. The configuration files corresponding to each defined pool are located inside the directory `/etc/libvirt/storage`. A wide variety of pools are supported, but the more important ones for the purpose of the present study are:

- Local directory or filesystem (including network filesystems such as NFS): if a filesystem is configured as a storage pool, Libvirt mounts it on a local directory to manipulate the files stored inside it (Libvirt, 2018). Once it is mounted, the pool is handled in the same manner as a local directory. Using this type of pool, Libvirt manages each virtual hard drive as a single file (Libvirt, 2018). For example, if a VM is created with one virtual hard drive, the corresponding file is also created inside the filesystem previously mounted as a storage pool. This means a separate file is generated per each virtual hard drive assigned to a VM.
- Physical disk pool: if a physical disk is configured as a storage pool, Libvirt manipulates the partition table and creates one partition per each virtual hard drive generated (Libvirt, 2018). This means a separate partition is generated inside the partition table of the physical disk per each virtual hard drive assigned to a VM.
- Logical volume pool: this type of pool uses the logical volume management (LVM) support provided by the Linux kernel, which allows the creation of logical volume groups from multiple physical partitions or entire hard drives. If a logical volume group is configured as a storage pool, Libvirt creates one logical volume inside the group per each virtual hard drive generated (Libvirt, 2018). This means once a logical volume group has been created, a separate logical volume is generated inside the volume group per each virtual hard drive assigned to a VM.

In summary, a virtual hard drive assigned to a VM can be created as a file in a mounted filesystem, as a physical partition in a physical disk, or as a logical volume in a

logical volume group. The storage pools based on physical partitions or logical volumes are called block-based storages because they use devices without a filesystem as backend to store virtual hard drives (RedHat, 2018). On the contrary, pools based on local directories or filesystems are called file-based storages and they do use a device with a filesystem as backend to store virtual hard drives (RedHat, 2018). Block-based storage pools offer a higher performance than file-based since VMs using block-based devices achieve higher throughput and lower latency (IBM, 2018).

The next concept to consider regarding storage management is the allocation policy. There are two possible options:

- Preallocated (thick provisioned) storage: this policy allocates all the storage space of a virtual hard drive at the moment it is generated (RedHat, 2015). This means if a VM is created with a 50GB virtual hard drive in a filesystem pool, then the corresponding file is created with a size of 50GB. This policy offers a better writing performance because no storage allocation occurs during runtime and it is recommended for VMs that works as servers or that need high I/O performance (RedHat, 2015). The main disadvantage is that all the storage is allocated even if only a small part of the virtual hard drive is being used.
- Sparsely allocated (thin provisioned) storage: this policy is more flexible than preallocated storage because the virtual hard drive grows as the VM writes data on it (RedHat, 2015). It also offers a lower writing performance and it is recommended for VMs that work as desktops or that do not need high I/O performance (Libvirt, 2018)

The last concept to take into account in storage management is the virtual hard drive format. Libvirt supports several formats: raw, cow, qcow, qcow2, qed, vmdk, and more (Chirammal et al., 2016). Even though several formats are supported (including proprietary ones such as vmdk), the best performance in a KVM virtualization node is achieved when the format used for the virtual hard drive is one of the following:

- raw: it supports both allocation policies (preallocated and sparsely allocated) when using a file-based storage pool as backend, and only preallocated policy when using

block-based storage pool (RedHat, 2017). This format is a direct representation of the virtual hard drive content and does not add additional metadata to it (Chirammal et al., 2016; RedHat, 2017).

- qcow2: it supports both allocation policies but only when using a file-based storage pool as backend. This format adds additional metadata to the content of the virtual hard drive and it also offers advanced features such as snapshots, compression, and encryption (RedHat, 2017).

The raw format has a better performance because no additional formatting is applied to virtual hard drive (RedHat, 2018). On the contrary, qcow2 offers a better snapshot support, but a similar behavior can be achieved using raw virtual hard drives in a logical volume pool (RedHat, 2017). In fact, the usage of logical volume pools to store preallocated VM hard drives in raw format is a convenient strategy for cloud providers because it combines the high performance offered by block-based storages and by the raw format with the flexibility of logical volumes.

Table 2.1 summarizes the relationship between the KVM storage components previously introduced and their corresponding support.

Table 2.1. Storage pool, allocation policy, and virtual hard drive format supported by KVM

Storage pool	Allocation policy	Virtual hard drive format	Supported
Block-based (physical disk or logical volume)	Preallocated	raw	Yes
		qcow2	No
	Sparsely allocated	raw	No
		qcow2	No
File-based (local directory or filesystem)	Preallocated	raw	Yes
		qcow2	Yes
	Sparsely allocated	raw	Yes
		qcow2	Yes

In a KVM virtualization node, the following command can be used to determine the format and allocation policy of an existent virtual hard drive identified by the last parameter: `qemu-img info diskfile` (Chirammal et al., 2016).

There is another concept related to virtual storage: the type of virtual hard drive assigned to the VM. The implementations supported by Libvirt are: IDE, SATA, SCSI, or VirtIO (between other options). From a digital forensic standpoint, the type of hard drive assigned to a VM is not relevant because the process to collect data from it is the same. However, it is worth mentioning that VirtIO is a paravirtualized implementation, while the others are just emulations. For this reason, VirtIO offers a higher performance and it is likely to be used in production environments.

2.2.2.2 Virtual storage snapshots and images

A snapshot represents the state of a VM hard drive in a particular point in time. Libvirt supports the creation of live snapshots that can be taken while the VM is running, but if the VM is generating high I/O traffic it is safer to stop or suspend the VM before taking the snapshot (Chirammal et al., 2016). The utility virsh allows the creation of virtual hard drive snapshots, regardless of whether the VM is running, suspended, or stopped. Two different type of snapshots can be taken:

- Internal: the snapshot is contained within the virtual hard drive of the VM. It is only supported by qcow2 virtual hard drives and does not support logical volume storage pools (Chirammal et al., 2016).
- External: the original virtual hard drive becomes read-only and the writing operations are completed on a new overlay file (Chirammal et al., 2016). It is supported by all the virtual disk formats, including raw and qcow2 (Chirammal et al., 2016). The new overlay file is created with qcow2 format and it can grow to the same size defined for the original virtual hard drive (Chirammal et al., 2016).

From a digital forensic perspective the external approach seems to be optimal because it does not modify the original virtual hard drive. Once the snapshot is created a bit-stream image can be created from the read-only hard drive. In addition, internal snapshots are supported by only one virtual hard drive format: qcow2.

As it has been stated, the utility virsh can be used to create external snapshots. For example, the next command creates an external snapshot of a VM named VM1: virsh

snapshot-create-as VM1 --disk-only --atomic --name snapshot1 --diskspec hda,snapshot=external,file=/mnt/kvm/VM1-snapshot.qcow2 . The parameter --diskspec defines three additional options: the virtual hard drive of the VM to set as read-only (hda), the type of snapshot (external), and the new overlay file to be created (/mnt/kvm/VM1-snapshot.qcow2). The parameter --disk-only specifies not to include the memory content in the snapshot. The parameter --atomic assures the snapshot either succeeds or fails with no changes on the original device. It is recommended to use --atomic every time an external snapshot is created, specially when it is created from a live system (Chirammal et al., 2016). Finally, the parameter --name specifies a customized name for the snapshot generated. Once the previous command is successful, it is possible to create a bit-stream image from original virtual hard drive, which is set as read-only after the overlay file was created.

Once the bit-stream image has been created and verified, the overlay file can be merged into the original virtual hard drive through the command: `virsh blockcommit VM1 hda --active --pivot --verbose`. In this example, the parameter `hda` specifies the virtual hard drive, the flag `--active` initiates the merging process of the overlay file into the virtual hard drive, and `--pivot` makes the hard drive active again once the merging is completed. At this point all the read/write operations are completed on the hard drive again and the overlay file is not used anymore. The flag `--verbose` displays detailed information about the process on the screen.

The previous command does not remove the overlay file nor the snapshot metadata maintained by Libvirt. The following command should be executed to remove the metadata: `virsh snapshot-delete VM1 snapshot1 --metadata`. The parameter `snapshot1` is the customized name specified for the snapshot previously generated. At the time of this study, Libvirt does not have support for removing external overlay files (Chirammal et al., 2016). For this reason, once the metadata is removed, the overlay file has to be manually deleted, for example in this particular case, by executing the command: `rm /mnt/kvm/VM1-snapshot.qcow2`.

2.2.2.3 Virtual RAM

In a KVM environment, each VM defines two key parameters regarding the RAM allocation policy used:

- **Maximum allocation:** it represents the maximum RAM that can be allocated to the VM when it is in execution.
- **Current allocation:** it represents the actual RAM allocated to the VM when it is powered on. It can be lower or equal to the maximum allocation value.

Each VM runs like a normal GNU/Linux process inside the host OS. When a VM is powered on, its process allocates the RAM specified by the parameter current allocation. If the VM needs more RAM, the allocation increases until the maximum allocation value (Chirammal et al., 2016). This behavior is achieved through a driver named balloon that allows each VM to notify the hypervisor how much RAM it requires (RedHat, 2017). The hypervisor then dynamically assigns available RAM between different VMs according to their needs (RedHat, 2017). Even though this is a flexible approach, it could lead to a potential problem: if the sum of the maximum allocation values of all the different VMs is greater than the physical RAM of the host and all the VMs request the maximum allocation at the same time, the host OS will become unstable. Therefore, it is recommended that the current allocation value equals the maximum allocation value and that the sum of these values for all the VMs is lower than the physical RAM available in the host.

2.2.2.4 Virtual RAM images

The utility `virsh` can be also used to create an image of the RAM content of a VM, but it has to be suspended in order to obtain a consistent result (Suneja, Isci, & de Lara, 2015). The command `virsh suspend` pauses the execution of the VM, but keeping its RAM allocation unmodified. Once the VM is suspended, the RAM content can be dumped to a local file through the command `virsh dump`. For example, the next command creates an image of the RAM content of a VM named VM1: `virsh dump VM1 --memory-only /mnt/kvm/VM1.memdump`. The resulting file is `/mnt/kvm/VM1.memdump` and the parameter `--memory-only` specifies to collect only the RAM content and CPU common register value of the VM. After the RAM image is created, it is possible to resume the

execution of the VM with the command `virsh resume`. It is important to note that the command `virsh dump` automatically suspends the VM before creating the RAM image and resumes it after the image is completed. In case the VM should not be suspended during a particular investigation, a RAM image can still be created if the flag `--live` is added to the `virsh dump` command. In this case the RAM image is created while the VM is running.

The command `virsh dump` generates a RAM image in QEMU ELF core dump format, which is supported by the Volatility Framework (Volatility, 2018). Using this utility, it is possible to examine the RAM image content looking for running processes, network connections, opened files, and other artifacts stored in RAM.

2.2.2.5 Virtual networking

In Libvirt, the key component of virtual networking is the Ethernet bridge, which simulates a virtual network switch where different network interfaces can be attached (Chirammal et al., 2016). By default, Libvirt defines only one virtual network, which creates an Ethernet bridge on the virtualization node named `virbr0` and with IP address `192.168.122.1`. The network interfaces of the VMs can be attached to this bridge (working as a network switch) in order to allow inbound and outbound network traffic between the VMs and with the virtualization node (RedHat, 2017). The configuration file of the default virtual network is `/etc/libvirt/qemu/networks/default.xml`.

Libvirt also creates one TAP interface on the virtualization node (named `vnet<number>`) per each network interface added to a VM (Chirammal et al., 2016). If an Ethernet bridge represents a network switch, these interfaces represent the ports of the switch where other devices are connected.

Figure 2.2 illustrates a network diagram of a virtual network with a bridge and two VMs attached to it. In this example, the virtualization node has two physical network interfaces (`eth0` and `eth1`) connected to different networks. It also has a bridge (`virbr0`) whose main function is working as a virtual switch with two virtual ports (`vnet0` and `vnet1`), where the network interfaces of the VMs are connected.

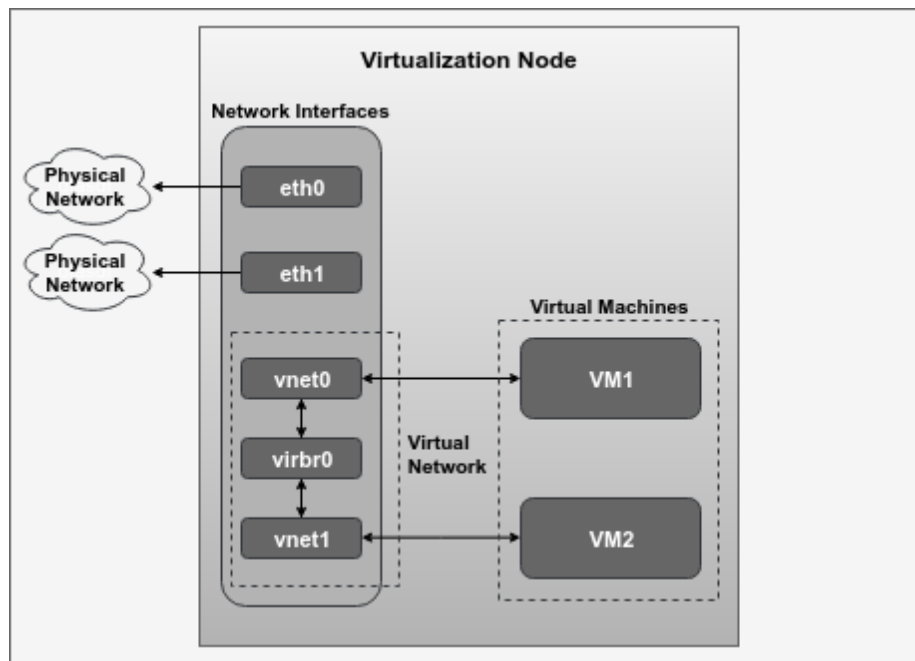


Figure 2.2. KVM virtual network diagram

Once the virtual network interfaces of each VM are connected to the bridge through the TAP interfaces, they can be set to one of the following modes:

- **Isolated mode:** inbound and outbound network traffic between the VMs and the virtualization node is allowed. However, traffic beyond the virtualization node is not allowed.
- **NATed mode:** inbound and outbound network traffic between the VMs and the virtualization node is allowed. Outbound traffic beyond the virtualization node is also allowed by network address translation (NAT) rules.
- **Routed mode:** inbound and outbound network traffic between the VMs and the virtualization node is allowed. Inbound and outbound traffic beyond the virtualization node requires modifying the routing tables of the devices involved.
- **Bridged mode:** the VMs are attached to an Ethernet bridge and a physical network interface of the virtualization node is also attached to the bridge. This configuration makes the VMs visible on the physical network, allowing inbound and outbound

traffic between the VMs and the devices connected to the physical network without the need of modifying the routing table of the devices involved.

Isolated, NATed, and routed modes are common in testing environments, while the bridged mode is common in production environments because it allows access to the VMs directly through their assigned IP addresses (Chirammal et al., 2016). This mode is used, for example, by cloud providers to offer VPSs with public IP addresses to host services accessible from the Internet.

There is another concept related to virtual networking: the type of virtual network interface connected to the VM. The implementations supported by Libvirt are: rtl3189 (Realtek chipset), e1000 (Intel chipset), or VirtIO. In a similar vein as hard drives, from a digital forensic standpoint, the type of virtual network interface assigned to a VM is not relevant because the process to collect data from it is the same. However, it is worth mentioning that VirtIO is a paravirtualized implementation, while the others are just emulations. For this reason, VirtIO offers a higher network performance and it is likely to be used in production environments.

2.2.2.6 Virtual network traffic capturing

Tcpdump is a powerful command line utility for monitoring and capturing network traffic in real-time. It captures the traffic bit-by-bit as it passes through any network interface and it can also decode protocols from layer 2 (data link), layer 3 (network), and layer 4 (transport) of the OSI model (Davidoff & Ham, 2012). The network capturing process performed by tcpdump is highly accurate and, in consequence, it is also considered admissible in a court of law (Davidoff & Ham, 2012). However, this accuracy may be limited by two factors. In the first place, capturing network traffic is a CPU-consuming activity, which means on high traffic networks the CPU could be overloaded and tcpdump could not be capable of capturing every packet, ignoring some of them (Davidoff & Ham, 2012). The second factor is the hard drive available space. If the capturing is too broad and it includes numerous devices generating high traffic operations, the available hard drive space in the capturing station could not be enough to store all the traffic (Davidoff & Ham, 2012). For these reasons, it is recommended to capture the

network traffic when the CPU load is low and to plan previously the amount of hard drive available space needed to store the data collected through this procedure (Davidoff & Ham, 2012). It is also crucial to filter the network traffic to be captured by specifying only particular hosts or ports related to the ongoing digital investigation. This determination leads to a lower CPU-consuming process and to a smaller capture file, which is also beneficial for the subsequent analysis. In addition, the live migration support provided by KVM could be employed to migrate a particular VM to a special virtualization node. The network capturing process could be completed on this node exclusively set aside and with enough resources to complete this task appropriately.

As it was previously mentioned, Libvirt leverages the bridge support provided by the Linux kernel to set up a virtual network environment for the VMs. This means tcpdump could be used to capture the network traffic that traverses a bridge (working as a network switch), and therefore, to capture all the network traffic of that virtual network. This is possible because the bridge is just a network interface created in the virtualization node. Nevertheless, this approach has two potential weaknesses from a digital forensic perspective. Firstly, it is not efficient and the accuracy of the process could be compromised as it was previously stated. Secondly, it could also affect the privacy of users, since the network traffic from all the VMs running in the virtualization node is captured, even if only one VM is under investigation. It seems much more appropriate to take advantage of the Libvirt virtual network management and use tcpdump to only capture the traffic that traverses the TAP interfaces of the interested VM.

By taking the previous analysis into consideration, a possible way to capture the network traffic of one single VM, whose TAP interface is vnet0, is through the command: `tcpdump -nn -s0 --interface=vnet0 -w /mnt/forensic-data/network-capture.pcap`. The parameter `--interface` specifies to capture only the traffic that traverses the listed network interfaces (vnet0). The parameter `-nn` declares not to resolve IP addresses (or ports) to host names (or services names). This option is relevant because the resolution process executed by default generates an important delay that can be omitted. The parameter `-s` indicates the snapshot length, which is the amount of bytes from each packet to be captured. In this case, the value 0 means no limit and it was included to avoid truncating

packets that have a longer size to the value specified. However, depending on the investigation and the legal constraints, this value could be reduced. For example, if the network is based on the Ethernet standard as data link layer, a value of 1,514 bytes should be enough because this is the maximum size of an Ethernet packet (Davidoff & Ham, 2012). Finally, the parameter -w declares the output file where the network traffic captured is stored. Tcpdump saves this file using the pcap (packet capture) format, which is an API for capturing network traffic. This format is also supported by network packet analyzers such as Wireshark or SolarWinds.

2.2.3 VPS

IaaS providers make use of virtualization technologies such as KVM by offering VPSs to their customers. In this context, a VPS is a synonym of a VM, with the particularity that it is managed by the customer but hosted in the cloud provider infrastructure (Barrett & Kipper, 2010). Figure 2.3 depicts the relationship between a KVM virtualization node and the VPSs hosted by it.

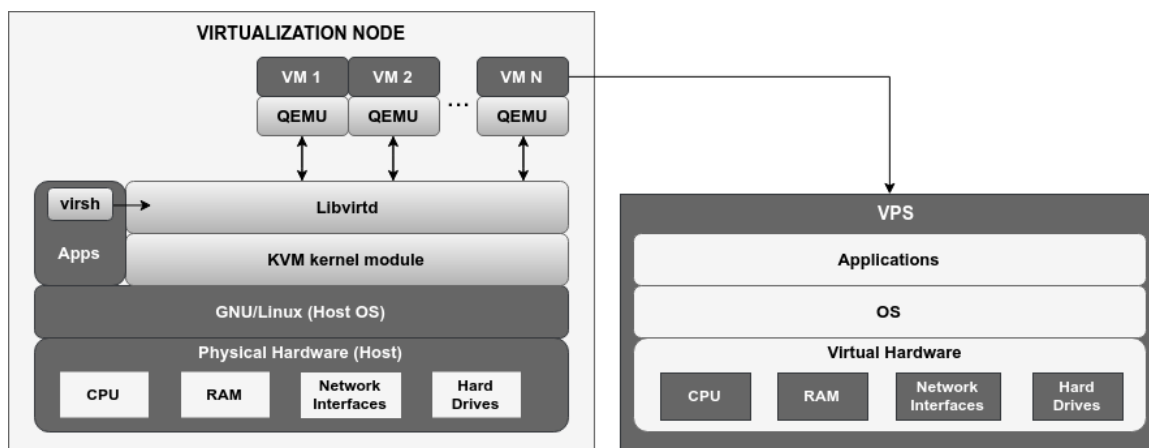


Figure 2.3. KVM virtualization node and VPSs overview

Customers pay according to the resources used and they have total control over their VPS (i.e., they can modify the operating system or install and manage any software they need). One of the most popular VPS solutions is Amazon EC2, which is provided by

Amazon Web Services (AWS), and whose cloud is formed by more than half-a-million GNU/Linux servers with XEN as a hypervisor (ZDNet, 2012). However, AWS recently announced a shift from Xen to KVM for future EC2 VPSs (TheRegister, 2017).

2.3 Cloud Computing and Virtualization in Digital Forensics

The research needs on cloud computing and virtual environments from a digital forensic point of view have constantly been evolving. For this reason, it seems appropriate to present in the first subsection the early studies that identified these two areas as new challenges to be investigated further. In the second subsection, more recent studies are analyzed from the IaaS perspective to recognize the improvements and unaddressed topics. The third subsection introduces studies on complementary approaches to improve the availability of digital evidence in the cloud provider environment. Finally, the fourth section analyzes the impact of using the virtual machine introspection (VMI) approach to acquire digital data from the hypervisor level.

2.3.1 Early Research

Pollitt et al. (2008) developed a research agenda to examine the impact of virtualization on digital forensics. The authors recognized three principal research areas to be developed further: analysis of virtual environments, virtualization as investigative tool, and virtualization in education. Three main sub-areas were identified inside analysis of virtual environments that needed to be addressed in order to conduct valid forensic investigations: forensic data acquisition of VMs, virtual platform forensics, and virtual machine introspection (Pollitt et al., 2008).

The positive use of virtualization with investigative and educational purposes was previously highlighted by Bem and Huebner (2007). They stated this technology was a significant step forward to reduce the time needed to analyze digital evidence and also to train new forensic analysts without compromising real evidence.

Beebe (2009) conducted a review study to examine the body of knowledge in digital forensics and to evaluate the improvements, issues, and unaddressed topics in the field. The most important contributions made by the forensic community were the progress toward formalizing the discipline and the improvements related to identifying, collecting, and examining digital data in traditional computer platforms (Beebe, 2009). However, the study acknowledges that achieving the same positive results on non-standard computing environments, such as cloud computing or virtualization, has not been properly addressed (Beebe, 2009). According to the author, research efforts had been made to take advantage of virtualization in education or investigative contexts, but there was still a lack of research on how to conduct digital forensic investigations in virtual environments. Another unaddressed topic mentioned was the increase of volume storage capacities and the importance to performing selective forensic acquisitions in order to simplify the analysis process (Beebe, 2009).

The previous studies suggest that before 2010 virtualization had been considered only as a tool but not as an environment to be inspected from a digital forensic standpoint. Similarly, there was no research on the impacts of cloud computing at all. These two facts implied important gaps in the digital forensic field that needed to be filled.

2.3.2 Cloud Computing and Virtualization

Dykstra and Sherman (2012) conducted a research study to expose technical and legal issues related to the acquisition of digital evidence from an IaaS cloud provider, and the strategies proposed to address these limitations. The authors identified six abstraction layers in the IaaS model (from lower to higher): network, physical hardware, host OS, hypervisor, VM OS, and VM applications. Each layer determines the type of data collected: network packets at the network layer, hard disk sectors at the physical hardware layer, or file system data at the host OS layer. At the same time, each layer offers a different level of trust regarding the quality of the data acquired. The lower the layer, the more reliable is the evidence and less level of trust is needed by judges or jury to consider admissible the evidence in a court of law (Dykstra & Sherman, 2012). If the hypervisor is

type I, then the third and forth layers (host OS and hypervisor) are combined in just one, and VMI utilities could be used to collect data directly from this layer, which is more reliable than the higher ones (Dykstra & Sherman, 2012).

Dykstra and Sherman (2012) also evaluated the capability of different forensic tools to collect data remotely from an Amazon EC2 server. FTK and EnCase were able to retrieve data from the VM OS (layer 5), but they required a local client installed on the VM and they did not provide hash values for integrity verification (Dykstra & Sherman, 2012). These are important limitations, because installing a local client is not a valid option when the VM being investigated was used to commit a crime. In this case, the VM should not be modified to not reveal the presence of the examiner and to not affect the integrity of the data to be collected. The authors also employed VMI techniques to inject the local client in the memory of the VM, avoiding the need of installing the local client. This approach allowed them to retrieve more reliable information from the hypervisor level (layer 4), but it leaves traces that could reveal the presence of the examiner.

According to Dykstra and Sherman (2012) there are also different options about who should perform the collection of the digital evidence: law enforcement, an employee of the cloud provider, or an independent examiner. In the majority of legal cases in the US, a search warrant or a subpoena is issued to the cloud provider, which executes the collection and then provides the data to the law enforcement agency (Dykstra & Sherman, 2012). This procedure releases law enforcement from performing the acquisition, but the process needed to justify the admissibility of the evidence becomes more complex because it relies on the experience and integrity of the technician at the provider (Dykstra & Sherman, 2012). This limitation could be partially addressed if a guideline or standard procedure is developed to be followed by the technicians of the cloud providers. This document should include the steps required not only to collect the data, but also to include methods to verify and validate the data acquired.

Birk (2011) conducted a study to analyze the technical challenges of digital forensic in cloud computing environments. According to the author, in case of an incident, IaaS offers much more information to be collected and used as digital evidence than PaaS and SaaS. One important advantage of IaaS and virtualization is the usage of snapshots,

which is supported by all popular hypervisors and allows the creation of an exact copy of the virtual hard drives and the RAM content of a VM (Birk, 2011). This feature also allows digital forensic practitioners to collect information of a VM without the need of turning it off. Another benefit mentioned of IaaS and virtualization is the capacity of using VMI utilities to observe and collect data of the VMs from the hypervisor level (Birk, 2011).

In a similar vein, Birk and Wegener (2011) defined additional technical challenges related to the IaaS model. They claimed that best practice guides about collection of digital data are usually outdated and there are no guides to complete this process in a cloud or virtual environment. If these guides were updated, it would be simpler to justify in a court of law the integrity and authenticity of the data retrieved and the chain of custody of the digital evidence presented (Birk & Wegener, 2011).

Zawoad and Hasan (2013) performed a meta-analysis of challenges, approaches, and problems in cloud computing forensics. One challenge they mentioned was forensic data acquisition, which they considered the most important step in an investigation because it involves the collection of the potential digital evidence (Zawoad & Hasan, 2013). The authors also noticed that IaaS offers some advantages over traditional computer forensics such as the creation of snapshots from the hypervisor level. According to them, most of the approaches proposed to overcome limitations in cloud computing depend on the cloud providers' willingness to modify their environments and to improve their readiness level (Zawoad & Hasan, 2013). Instead, other solutions proposed are focused on the legal aspects of cloud computing by proposing an international legislation for cloud forensic investigations (Zawoad & Hasan, 2013).

Important results can be highlighted from the studies of Birk (2011) and Zawoad and Hasan (2013). First, the data to be collected from a cloud environment depends on the type of cloud model. For SaaS and PaaS the possible acquisition approaches are limited and rely on the provider's assistance and readiness. For IaaS, the control is higher because a VM could be accessed to collect data or even retrieve it from the hypervisor level, although the provider's collaboration is also needed. As it was stated before, a VPS is just a VM hosted in a IaaS provider's infrastructure, this means it is possible to create VM

snapshots and to get an exact copy of its virtual hard drives and RAM content. This is a more auspicious scenario than a traditional computer forensic investigation because it is not possible to create such snapshots from physical computers. Second, most of the solutions proposed for the current problems in this field were related to modifications to be accomplished by law makers or cloud providers in order to make the investigation process more viable. Third, the absence of guidelines was mentioned again which emphasizes the significance of defining practical procedures to complete digital forensic acquisitions in cloud and virtual environments.

Morioka and Sharbaf (2016) conducted a research on cloud forensic to identify unaddressed challenges and possible solutions. The main challenges they mentioned were the data acquisition difficulties, the privacy and confidentiality concerns when data is collected from shared infrastructure (e.g., virtual environments), and the cloud provider's lack of staff and procedures to conduct forensic investigations. They also identified three locations where digital information could be collected: local computers, between the cloud and local computers, and in the cloud. The first location refers to the personal devices used to connect to the cloud services, browser cache and history is a clear example of possible information to be found. The second location concerns records of the Internet service provider (ISP) which could contain traces of client-cloud communication. The third location refers to evidence directly located at the cloud provider. The authors mentioned different possible approaches to collect information from this latter location: data acquisition from physical node directly at cloud provider, virtual machine introspection (VMI) to monitor and examine VMs from the hypervisor level, and forensic tools to acquire information directly from the VM (Morioka & Sharbaf, 2016).

The focus of this study is data acquisition of VPSs (i.e., VMs hosted in a virtualization node owned by cloud computing provider), and for this reason the most fertile location to retrieve evidence is the cloud provider's infrastructure. However, performing a full data acquisition of the virtualization node is not a valid approach for two reasons: the volume of information to collect and analyze could be enormous and the privacy of the customers could be compromised. Using forensic tools to collect data directly from the VM is not a solid method either because it requires accessing the VM in

order to install and execute the tools (i.e., modifying the state of the VM). Therefore, the scope of the acquisition has to be reduced to include only the suspicious VPSs in order to not infringe on privacy rights of other clients. The optimal method to do this seems to be virtual machine introspection (VMI) which is the process of monitoring and managing VMs externally from the hypervisor level. By using this method, the presence of the forensic examiner is not revealed and the state of the VM is not modified by installing tools and executing them.

2.3.3 Complementary Approaches

Patrascu and Patriciu (2014) presented an alternative approach to monitor and log user activity in IaaS cloud environments. They proposed the idea of implementing a cloud forensic module to gather forensic and log data from the VMs running in a cloud provider. The forensic module was designed to be scalable in order to avoid performance issues and to be applied on top of new or existing cloud computing deployments (Patrascu & Patriciu, 2014). Their study focused on KVM as an example of the integration between the proposed module and one particular hypervisor. However, in a real case scenario, this integration has to be developed for any other hypervisor used by the cloud provider (Patrascu & Patriciu, 2014). They concluded the presented framework allowed digital investigators to collect a great deal of information from any particular VM in a cloud provider, no matter the heterogeneity of the VMs and geographical distribution of the cloud (Patrascu & Patriciu, 2014).

A similar strategy to deal with cloud environments in a digital investigation was proposed by Sang (2013), who recommended to build a forensic-friendly environment in cloud providers. This model focused mainly on SaaS and PaaS solutions and it should be implemented by the cloud providers to keep the log records of the interaction between users and the cloud services (Sang, 2013). According to the author, the cloud provider should implement mechanisms to make this information secure and available not only on its side, but also on the customer's side.

The methods proposed by Patrascu and Patriciu (2014) and Sang (2013) provided recommended practices to be performed on the cloud service provider's side as a readiness strategy to increase the amount of potential evidence available. Both methods could be helpful in an investigation by providing additional data, but the downside is that their implementation depends on the cloud provider's willingness to cooperate. From a digital forensic perspective it is fundamental to have standard and reliable procedures to perform a comprehensive acquisition in virtual environments, regardless of whether the cloud provider contributes with additional information from its end.

2.3.4 Virtual Machine Introspection

Virtual Machine Introspection (VMI) is a method that allows to monitor and control the state of the VMs from the hypervisor level. Through this approach, the hypervisor can inspect all the data processed by the VMs because it is located in the layer between the physical hardware of the virtualization node and the VMs (Poisel, Malzer, & Tjoa, 2013). From this layer, the hypervisor can examine the virtual resources of a VM (i.e., hard drives, RAM content, and network traffic) without altering its state. This method allows investigators to overcome the classical limitations presented in traditional computer forensics such as RAM and network traffic acquisitions (Poisel et al., 2013). VMI tools relies on the following principles: the VMs are unable to interfere with the operations executed at the hypervisor level (i.e., tamper resistant), the hypervisor has total control over the state of the VMs (i.e., evasion resistant), and the hypervisor has a complete and direct access to the VM resources such as CPU registers, RAM content, and network traffic (i.e., efficiency) (Hebbal et al., 2015).

The simplest way to retrieve data from a VM using VMI utilities is in binary format (e.g., a RAM dump or a virtual hard drive image). Other VMI tools have been developed to provide a high-level information such as processes running in the VM instead of just bits. The difference between bits and high-level information is called the semantic gap (Dolan-Gavitt, Payne, & Lee, 2011; Hebbal et al., 2015; Poisel et al., 2013). According to the method used by the VMI utilities to monitor the state of the VMs,

they can be classified in: in-VM (or in-brand), out-of-VM (or out-of-brand), and hybrid (or derivation) (Hebbal et al., 2015; Poisel et al., 2013). The in-VM strategy implies installing an agent inside the VM in order to expose its state to the hypervisor. On the contrary, out-of-VM is considered the real VMI strategy because it does not need an agent inside the VM, it collects the data from the hypervisor level by accessing the VM resources directly. The hybrid strategy is a combination of in-VM and out-of-VM. There are also VMI utilities that can be available out-of-the-box on a particular hypervisor, and others that need to modify the default behavior of the hypervisor or install additional libraries in order to be able to work (Suneja et al., 2015). The client `virsh` previously mentioned in subsection 2.2.2 is an example of an out-of-VM and out-of-the-box VMI utility that can be used to monitor the state of the VMs in a virtualization node that uses KVM as a hypervisor (Suneja et al., 2015).

According to the previous observations, the VMI out-of-VM and out-of-the-box strategy is the best option from a digital forensic perspective because it does not need to alter in any manner the content of the VMs or the hypervisor in order to collect reliable data. Through these specific VMI utilities, it could be possible to create an image of the virtual hard drive, dump the RAM content, or capture the network traffic in real time. In addition, they use the inherent benefits of virtualization, such as creating snapshots or suspending the VMs. The snapshot feature permits to create an instant copy of the virtual hard drive at a particular point in time, without the need of stopping or suspending the VM. This is also beneficial to be undetected by the user (customer) while the acquisition is being performed. The capability of suspending the VMs allows the investigator to create consistent RAM content images and to validate their integrity (Poisel et al., 2013). This is conceivable because once the VM is suspended the RAM content remains unmodified and it can be imaged from the hypervisor level in a reliable and efficient manner. The RAM image created can be analyzed using memory forensic utilities such as the Volatility Framework (Volatility, 2018). The drawback of suspending the VM is that it could be detected by the user because the VM is not accessible until it is unsuspended. Through these VMI utilities, it could be also possible to capture all the network traffic sent and received by a VM during a particular period of time and store it into a file. This file can be

inspected to analyze the network communications established using network analysis tools such as Wireshark (WireShark, 2018). The available space on the hard drive needed to capture the network traffic depends on the period of time the capture is active and on the volume of the network data sent and received by the VM during that time. In addition, this process may require an intensive usage of CPU on the hypervisor because each single network packet is first written to a file and then sent to the virtual network interface.

2.4 Summary

The following conclusions can be drawn from the previous analysis. First, the acquisition process on cloud computing and virtual environments is mentioned as an unaddressed topic in digital forensics (Beebe, 2009; Birk & Wegener, 2011; Morioka & Sharbaf, 2016; Pollitt et al., 2008; Zawoad & Hasan, 2013). Second, some of the solutions proposed to address this topic need active participation of the cloud providers in order to apply important modifications to their current environments (Patrascu & Patriciu, 2014; Sang, 2013). Third, VMI is introduced as a reliable approach to collect data from a VM because it works at a hypervisor level (Birk, 2011; Dykstra & Sherman, 2012; Morioka & Sharbaf, 2016). This method can be considered a selective acquisition because it only collects data from a particular VM instead of from the entire physical virtualization node. Therefore, it helps to address the issues of analyzing enormous data storage volumes and compromising the privacy of other customers (Beebe, 2009; Morioka & Sharbaf, 2016). In addition, the capacity to take snapshots and suspend VMs are mentioned as significant and positive advantages of collecting data from a virtual environment compared to traditional physical computers (Birk, 2011; Chirammal et al., 2016; Poisel et al., 2013). Fourth, one limitation noticed is the lack of updated guidelines to provide forensic investigators with reliable procedures to be followed in cloud or virtual environments (Birk & Wegener, 2011; Dykstra & Sherman, 2012). This limitation negatively affects the reliability of the evidence acquired from these environments. Fifth, there are different types of VMI utilities, but the out-of-VM and out-of-the-box ones are the more interesting from the digital forensic perspective because they do not need to alter

in any manner the content of the VM or the hypervisor. In spite of this fact, there is no research on the reliability of this kind of VMI utilities to collect data from a virtual environment. Instead, most of the studies focused on possible classifications and applications of diverse VMI tools (Hebbal et al., 2015; Poisel et al., 2013; Suneja et al., 2015).

2.5 This Research Study

This study took a different approach. It aimed to examine if it was possible to acquire forensically-sound digital evidence from a VPS hosted in a virtualization node that uses KVM as a hypervisor. To this purpose, the performance of two utilities was evaluated. The first one was virsh, a VMI out-of-VM and out-of-the-box utility, which was used to create images of the virtual hard drives and the RAM content of the VPSs. The second utility was tcpdump, which was used to capture in real-time the network traffic of the VPSs. Even though tcpdump has been studied and used in the digital forensics field to capture network traffic on physical network interfaces for a long time, this study focused on using this utility to capture traffic on specific virtual network interfaces assigned to a VPS. The main reason for focusing on these utilities was they could positively impact on the admissibility of the evidence in a court of law because virsh works at the hypervisor level and tcpdump at the virtualization node OS level. This means the results of these utilities are more reliable than other utilities that work at the VM level (Dykstra & Sherman, 2012). Furthermore, they are available in the majority of the virtualization nodes based on GNU/Linux distributions and their installation and execution do not require significant modifications to the production environment.

CHAPTER 3. METHODOLOGY

As discussed in the previous chapters, there is a need in the digital forensics field to find an efficient and reliable procedure to acquire digital data from VPSs. This study aimed to analyze if the utilities `virsh` and `tcpdump` can be used to accomplish this objective in a full virtualized environment that uses KVM as a hypervisor. The principal reasons for focusing on these tools were: they produce more reliable results than utilities that work at the VM level (`virsh` works at the hypervisor level and `tcpdump` at the virtualization node OS level), they are available in the majority of the virtualization nodes based on GNU/Linux distributions, and their installation and execution do not require significant modifications to the production environment. This chapter introduces the research question, hypotheses, participants, research design, testing conditions and procedures, measurements for evaluation and success, and possible threats to validity.

3.1 Research Question and Hypotheses

The principal objective of this research was to answer the following research question: is it possible to acquire forensically-sound digital evidence from a VPS hosted in a cloud provider's virtualization node that uses KVM as a hypervisor?

The hypotheses of this study were stated as follows:

H₁: images of the hard drives of a VPS can be created in an efficient manner, respecting the integrity and completeness of the data acquired.

H₂: images of the RAM content of a VPS can be created in an efficient manner, respecting the integrity and completeness of the data acquired.

H₃: the network traffic of a VPS can be captured in real-time, respecting the integrity and completeness of the data acquired.

The utility `virsh` was used to test H₁ and H₂, while `tcpdump` was used to test H₃.

3.2 Participants

Five participants cooperated to address H_2 and H_3 by following four different scripts to interact with the research environment described in section 3.3. The participants selected were students of the Computer and Information Technology department at Purdue University because they are technically skilled and they could follow the scripts more naturally. The participants did not provide any personal information or opinions during the process and the purpose of this interaction is explained in subsection 3.5.2. Appendix F includes the review exemption letter provided by Purdue's Institutional Review Board (IRB) for this research.

3.3 Research Design

A study was conducted to test the hypotheses and to address the research question stated in section 3.1. The study involved the creation of a virtual environment to simulate a VPS provider's infrastructure. One physical computer, one Ethernet switch, and one Ethernet patch cord were used to this purpose. The computer acted as a KVM virtualization node and was connected to the switch (using the patch cord) to imitate the VPS provider's network. The computer required one network interface and enough hard drive and RAM resources to host four VMs. It also required two hard drives: one to store the VM hard drives and a second one to store the information collected from the VMs through the utilities *virsh* and *tcpdump*. These VMs acted as independent VPSs hired by different customers.

In addition, one laptop and another Ethernet patch cord were required for this study. The laptop was also connected to the switch (using the second patch cord) to simulate the network communication between the customers (or other external users) and the VPSs. Figure 3.1 summarizes the research environment proposed.

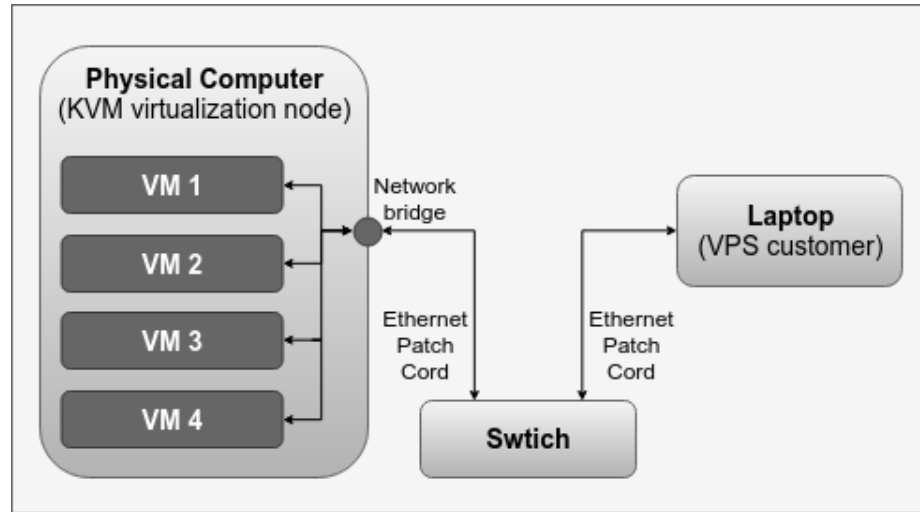


Figure 3.1. Research environment

Once the research environment was prepared, a BASH script was developed to address H_1 by testing if the utility `virsh` was able to take snapshots of the hard drives of the VMs, from where bit-stream images could be created. This BASH script was executed once per day, during 14 consecutive days, and after each execution the results were evaluated in terms of efficiency, integrity, and completeness. The source code of the BASH script was included in Appendix E. When the previous analysis was over, five different participants interacted twice with the four VMs (one at a time) using the laptop in order to address H_2 and H_3 . During each interaction all the network traffic was captured with the utility `tcpdump`, and at the end of the interaction an image of the RAM content of the VM was created with the utility `virsh`. Each RAM image creation process was evaluated in terms of efficiency, integrity, and completeness while each network traffic capturing process only in terms of integrity and completeness. The details of these processes are described later in this chapter.

The following two subsections describe the technical specifications of the devices used in the study and the installation and configuration process followed for each of them.

3.3.1 Hardware Specifications

This subsection details the technical specifications of all the devices used to recreate the research environment previously presented.

3.3.1.1 Virtualization node

The physical computer used as virtualization node in the study was a MacPro3,1 server, model A1186, with the following hardware specifications:

- CPU: Two 3.0GHz 45-nm Intel Xeon E5472 (Harpertown/Penryn) 64-bit processors. Each processor has four cores and a 12MB level-2 cache. These processors include the VT-x (virtualization technology) extension, which offers support for hardware-assisted virtualization.
- RAM: 32GB. Eight 4GB 800MHz DDR2 FB-DIMMs connected into the slots 1 to 4 on the first memory card and into the slots 5 to 8 on the second memory card.
- Hard Drive 1: Hitachi GST Deskstar HDT725040VLA360 3.5 inch internal hard drive, 400GB (physical sector size 512 bytes), 7200 RPM, SATA 3.0, 16MB cache. It was connected to the first 3.0 Gbit/s SATA bus connector of the motherboard. This hard drive was assigned to install the OS of the virtualization node and to store the VM hard drives.
- Hard Drive 2: Seagate BarraCuda ST31000333AS 3.5 inch internal hard drive, 1TB (physical sector size 512 bytes), 7200 RPM, SATA 3.0, 32MB cache. It was connected to the second 3.0 Gbit/s SATA bus connector of the motherboard. This additional hard drive was assigned to store the data collected from the VMs during the testing procedures.
- Video: ATI Radeon HD4870 1GB PCI-Express video graphics card.
- Ethernet Interfaces: Two independent Intel 10/100/1000 BASE-T Gigabit Ethernet RJ-45 interfaces.

3.3.1.2 Laptop

The laptop selected to interact with the VMs was a Lenovo IdeaPad Yoga 13, model 20175, Windows 8.1 Pro 64-bit, with the following hardware specifications:

- CPU: One 2.0GHz Intel i7-3537U 64-bit dual core processor. Each core has their own 0.25MB level-2 cache and 2MB level-3 cache.
- RAM: 8GB. One 8GB 1600MHz DDR3 SO-DIMM.
- Hard Drive: Samsung MZMTD256HAGM-000L1 internal solid state drive (SSD), 256GB (physical sector size 512 bytes), Mini-SATA (mSATA) interface.
- Video: integrated Intel HD Graphics 4000.
- Ethernet Interface: The laptop did not have an integrated Ethernet network interface. For this reason, a FosPower USB 3.0 to Gigabit Ethernet RJ-45 adapter was used to overcome this limitation and to connect the laptop to the switch.

3.3.1.3 Switch and Ethernet patch cords

Two category 6 Ethernet RJ-45 patch cords were used to connect the virtualization node and the laptop to a D-Link DGS-108 8-port Gigabit switch in order to simulate the cloud provider's network. Since the network interfaces, patch cords, and switch were Gigabit capable, the resulting Ethernet network supported a maximum throughput of 1 Gbit/s.

3.3.2 Virtualization Node Implementation

This subsection describes the process followed to install the OS and the KVM components on the virtualization node.

3.3.2.1 Operating system installation

The Ubuntu Server 16.04.4 64-bit LTS (long term support) version was downloaded from the official Ubuntu website. The latest LTS version was selected

because it is designed to be stable and supported for five years, while regular versions (non-LTS) are supported only for nine months. LTS is the version recommended for enterprises and for this purpose was used as OS of the virtualization node.

The downloaded file was named `ubuntu-16.04.4-server-amd64.iso`, with MD5 checksum `6a7f31eb125a0b2908cf2333d7777c82`. A bootable USB stick was generated with this file and the physical computer was booted from it. The option `Install Ubuntu Server` was selected, and each step of the installation process was completed as detailed below:

- Language: English
- Location: United States
- Keyboard layout (manually selected): English (US)
- Network configuration: Do not configure the network at this time
- Hostname: `kvm`
- Full name for the new user: `adolfo`
- User name for your account: `adolfo`
- Password for the new user: `m5LcbpygPRWHFa38`
- Encrypt your home directory: No
- Time zone: `America/Indiana/Indianapolis`
- Partition disks (manually selected): In this step, only the 400GB hard drive (Hitachi GST Deskstar HDT725040VLA360) was partitioned. This device was detected as `/dev/sda` by the Ubuntu installer. A new GUID partition table (GPT) was created, with four primary partitions. Table 3.1 shows the details of each partition defined.

Table 3.1. Partition table of device /dev/sda

Partition	Start Sector	End Sector	Size	Filesystem	Function
/dev/sda1	2,048	1,050,623	536.87MB	FAT32	EFI system
/dev/sda2	1,050,624	98,707,455	50GB	EXT4	Root partition
/dev/sda3	98,707,456	118,239,231	10GB	SWAP	Swap area
/dev/sda4	118,239,232	781,422,591	339.5GB	unformatted	Not used

Partition /dev/sda1 (EFI system) contained the UEFI files needed to start the loading process of the OS. Partition /dev/sda2 (root partition) contained all OS files and its mount point was the directory /. Partition /dev/sda3 was defined as swap space. Partition /dev/sda4 was created but not formatted during the Ubuntu Server installation process. This partition was used after the installation to calculate the transfer rate for the hard drive and then to store the VM hard drives. These two additional procedures are described later in this section.

- Updating policy: No automatic updates
- Software selection: Only the option Standard system utilities was selected to be installed.
- Install GRUB boot loader to the master boot record of the first hard drive: Yes (in this case the first hard drive was /dev/sda)
- Finish the installation: Continue (to boot into the new installed system)

Once the installation process was completed, the virtualization node booted into Ubuntu Server 16.04.4. The 400GB hard drive (Hitachi GST Deskstar HDT725040VLA360) was detected as device /dev/sda, while the 1TB hard drive (Seagate BarraCuda ST31000333AS) was recognized as /dev/sdb. The two integrated Ethernet interfaces were identified as enp7s0f0 and enp7s0f1. The interface enp7s0f0 was connected to a local network and configured to get Internet access. The interface enp7s0f1 was not enabled since it was not needed throughout the study. After that, all the installed

packages were updated to the latest version by executing as root the command: `apt-get update && apt-get upgrade && apt-get dist-upgrade`.

3.3.2.2 KVM installation and configuration

The KVM module was already installed by the `linux-image-generic` package (version 4.4.0-116.140). The KVM user-space tools (version 2.5.0), the `libvirt` service and client packages (version 1.3.1-1), and all the required dependencies were downloaded and installed by executing as root the command: `apt-get install kvm qemu-kvm libvirt-bin`. After that, the command `kvm-ok` (without parameters) was executed to determine if hardware-assisted full virtualization was supported by the CPU. The output of this command confirmed that it was supported and enabled.

3.3.2.3 Additional configuration steps

The following steps were performed to complete the configuration process of the virtualization node.

1. Hard drive `/dev/sdb` partitioning: the hard drive `/dev/sdb` was partitioned to replicate the same partition table defined for `/dev/sda` during the Ubuntu Server installation process. The principal objective of this replication was to define exactly the same fourth partition, which was used to calculate the transfer rates for both hard drives in subsection 3.4.1. In addition, the second partition was defined to have 50GB, which was the same size specified for the hard drives assigned to the four VMs. This partition was employed by a BASH script to create bit-stream images from the snapshots taken by the utility `virsh` in section 3.5. The other two partitions and the unpartitioned space available on `/dev/sdb` were not used during this study. Figure 3.2 illustrates (not to scale) the partition layout for both hard drives.

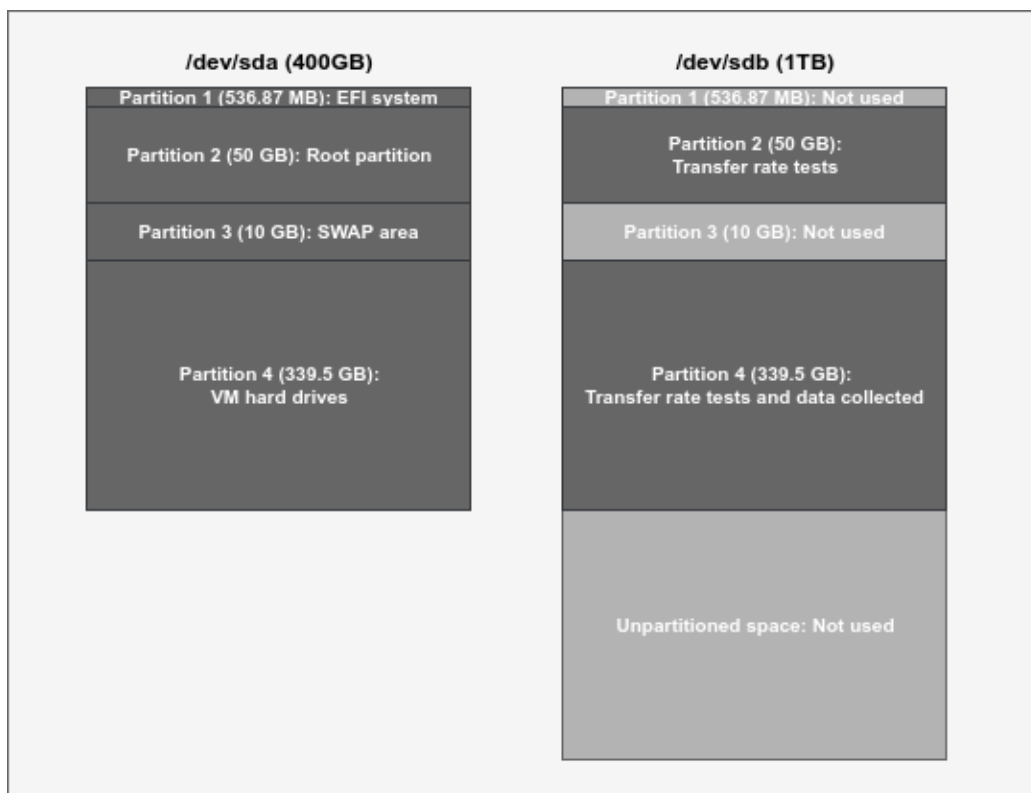


Figure 3.2. Partition layout for /dev/sda and /dev/sdb

2. Testing Conditions: once the hard drive /dev/sdb was partitioned, the procedures detailed in section 3.4 were completed. These procedures included: the examination of both hard drives for bad sectors, the calculation of read and write transfer rate for both hard drives, the data duplication transfer rate from /dev/sda to /dev/sdb, and the RAM data duplication transfer rate.
3. Partition /dev/sdb4 formatting: after the procedures detailed in section 3.4 were completed, the partition /dev/sdb4 was formatted with the EXT4 filesystem by executing as root the command: `mkfs.ext4 /dev/sdb4`. After that, the partition was mounted on the directory /mnt/forensic-data (previously created) by executing as root: `mount -t ext4 /dev/sdb4 /mnt/forensic-data`. The file /etc/fstab was also modified to mount this partition on boot in case the server was restarted.

4. Logical volume group creation: as it was stated in 2.2.2.1, the usage of logical volume pools to store preallocated VM hard drives in raw format is a convenient strategy for cloud providers because it combines the high performance offered by block-based storages and by the raw format with the flexibility of logical volumes. For this reason, a logical volume group was created in the virtualization node by executing two commands as root. The first one was: `pvccreate /dev/sda4`. It initialized the partition for use by the logical volume manager. The second command was: `vgcreate lvm-group /dev/sda4`. It generated a new logical volume group called `lvm-group` using the partition `/dev/sda4` as physical backend. This logical volume group was created to be used as storage pool for the VM hard drives.
5. Ethernet bridge configuration: as it was also mentioned in 2.2.2.5, the Ethernet bridge is the key component of virtual networking in KVM. It simulates to be a virtual network switch where different network interfaces can be attached. For this reason, an Ethernet bridge was created in the virtualization node by editing the file `/etc/network/interfaces`. This file was modified in order to reconfigure the Ethernet interface `enp7s0f0` and accomplish three objectives: disabling the Internet access previously configured, creating an Ethernet bridge named `br0` with IP address `192.168.10.1`, and attaching the interface `enp7s0f0` to the bridge. After the four VMs were created, their virtual network interfaces were set to bridged mode and attached to the bridge in order to share the same physical network with the virtualization node. Once the bridge configuration was finished, an Ethernet patch cord was used to connect the interface `enp7s0f0` to the physical switch to imitate the cloud provider's network.

Appendix A shows a complete list of all the packages installed on the virtualization node, including their respective version. Appendix B displays the customized content of the following configuration files: `/etc/fstab`, `/etc/network/interfaces`, and `/etc/libvirt/storage/lvm-group.xml`.

3.3.3 Virtual Machine Deployment

Four VMs were created in the virtualization node according the specifications detailed in Table 3.2. Different versions of OS and architecture were used to verify if the results were consistent across them.

Table 3.2. Virtual machine specifications

VM ID	Operating System	Architecture	Hard Drive	RAM	IP address
VM1	Ubuntu Server 17.10	32-bit	50GB	4096MB	192.168.10.11
VM2	CentOS 7	64-bit	50GB	4096MB	192.168.10.12
VM3	Windows Server 2008	32-bit	50GB	4096MB	192.168.10.13
VM4	Windows Server 2016	64-bit	50GB	4096MB	192.168.10.14

Four logical volumes of 50GB (one per each VM) were created in the logical volume group `lvm-group` previously defined in the virtualization node. The command executed as root to create the logical volume for VM1 was: `lvcreate -L 50000000000B -n VM1_ubuntu lvm-group`. The parameter `-L` specifies the size in bytes of the volume, the parameter `-n` defines the name of the volume, and the last parameter indicates in which logical volume group the new volume is created. Once the previous command was executed, the new logical volume was created as `/dev/lvm-group/VM1_ubuntu`. Other three volumes were created in the same fashion: `/dev/lvm-group/VM2_centos` (for VM2), `/dev/lvm-group/VM3_win2008` (for VM3), and `/dev/lvm-group/VM4_win2016` (for VM4).

The OS installation process in each VM was completed using the original ISO files provided by Ubuntu, CentOS, and Microsoft, respectively. Table 3.3 shows the files used in this study and their MD5 checksums. These files were downloaded in the virtualization node inside the directory `/mnt/kvm/isos`.

Table 3.3. ISO files used to install the OS in each VM

VM ID	File name and MD5 checksum
VM1	ubuntu-17.10.1-server-i386.iso MD5 checksum: f713724032a1b0fdbf3ebd90d2eec8d8
VM2	CentOS-7-x86_64-Minimal-1708.iso MD5 checksum: 5848f2fd31c7acf3811ad88eaca6f4aa
VM3	6001.18000.080118-1840_x86fre_Server_en-us-KRMSFRE_EN_DVD.iso MD5 checksum: 89fbc4c7baafc0b0c05f0fa32c192a17
VM4	14393.0.161119-1705.RS1_REFRESH_SERVER_EVAL_X64FRE_EN-US.iso MD5 checksum: 70721288bbcdfc3239d8f8c0fae55f1f

The OS installation process followed for each VM and their particular settings are detailed in the next five subsections.

3.3.3.1 VM1 - Ubuntu Server 17.10 (32-bit)

Virtual resources allocated to this VM:

- CPU: 8 cores.
- RAM: 4096MB.
- Hard drive: /dev/lvm-group/VM1_ubuntu (50GB).
- Network: virtual interface vnet0 attached to the bridge br0.

The previous resources and other information required to create the VM in the virtualization node were included in the file /mnt/kvm/xmls/VM1.xml. After that, the following command was executed to create the VM taking as input this file: `virsh define /mnt/kvm/xmls/VM1.xml`. The content of the file is included in Appendix C.

Once the VM was created, it was booted from the ISO file downloaded from the Ubuntu website to start the installation process. The hard drive of the VM was detected as /dev/sda by the Ubuntu installer. A new MSDOS partition table was created with two primary partitions: /dev/sda1 (46GB) and /dev/sda2 (4GB). The former partition was formatted with EXT4 filesystem. It contained all the OS files and its mount point was the

directory /. The latter partition was defined as swap space. At the software selection screen, only the option SSH server was selected to be installed. In addition, only one user account was generated and granted with sudo privileges:

- User name: cflstudent
- Password: 2hLMmVGt

When the installation process was completed, the VM was booted into the new installed Ubuntu Server 17.10. The user account was used to login into the system and configure the network interface with the IP address 192.168.10.11 (network mask 255.255.255.0, no default gateway, and no DNS servers). The SSH server was enabled by default to listen for incoming SSH connections on port TCP 22. The Apache web server (version 2.4.27-2) was installed by executing the command: `apt-get install apache2`. The default configuration was used, which means the web server listened for incoming HTTP connections on port TCP 80 and served the default HTML file `/var/www/html/index.html`.

In summary, two network services were installed and configured in this VM:

- SSH (port TCP 22): Secure Shell access allowed only for user cflstudent.
- Apache (port TCP 80): HTTP access allowed for everyone.

3.3.3.2 VM2 - CentOS 7 (64-bit)

Virtual resources allocated to this VM:

- CPU: 8 cores.
- RAM: 4096MB.
- Hard drive: `/dev/lvm-group/VM2.centos` (50GB).
- Network: virtual interface `vnet1` attached to the bridge `br0`.

The previous resources and other information required to create the VM in the virtualization node were included in the file `/mnt/kvm/xmls/VM2.xml`. After that, the

following command was executed to create the VM taking as input this file: `virsh define /mnt/kvm/xmls/VM2.xml`. The content of the file is included in Appendix C.

Once the VM was created, it was booted from the ISO file downloaded from the Centos website to start the installation process. The hard drive of the VM was detected as `/dev/sda` by the Centos installer. A new MSDOS partition table was created with two primary partitions: `/dev/sda1` (46GB) and `/dev/sda2` (4GB). The former partition was formatted with XFS filesystem. It contained all the OS files and its mount point was the directory `/`. The latter partition was defined as swap space. In addition, only one user account was generated and granted with sudo privileges:

- User name: `cf1student`
- Password: `2hLMmVGt`

When the installation process was completed, the VM was booted into the new installed Centos 7. The user account was used to login into the system and configure the network interface with the IP address 192.168.10.12 (network mask 255.255.255.0, no default gateway, and no DNS servers). The SSH server was installed and enabled by default by the Centos installer to listen for incoming SSH connections on port TCP 22. The vsftpd FTP server (version 3.0.2) was installed by executing the command: `yum install vsftpd`. The default configuration was used, which means vsftpd listened for incoming FTP connections on port TCP 21 and allowed access to the user's home directory files. One additional modification was performed in the vsftpd configuration file in order to support the passive data transfer mode through the TCP port range 10090-10100. Once installed and configured, vsftpd was enabled to automatically start on boot by executing the command: `systemctl enable vsftpd`. The firewall installed by Centos was modified to allow incoming FTP connections to the port TCP 21 by executing the command: `firewall-cmd --permanent --add-port=21/tcp`. A similar modification was performed to allow incoming data transfer connections to the passive port range by executing the command: `firewall-cmd --permanent --add-port=10090-10100/tcp`. Finally, the firewall rules were reloaded through the command: `firewall-cmd --reload`.

In summary, two network services were installed and configured in this VM:

- SSH (port TCP 22): Secure Shell access allowed only for user cflstudent.
- vsftpd (port TCP 21): FTP access allowed only for user cflstudent.

3.3.3.3 VM3 - Windows Server 2008 Standard (32-bit)

Virtual resources allocated to this VM:

- CPU: 8 cores.
- RAM: 4096MB.
- Hard drive: /dev/lvm-group/VM3_win2008 (50GB).
- Network: virtual interface vnet2 attached to the bridge br0.

The previous resources and other information required to create the VM in the virtualization node were included in the file /mnt/kvm/xmls/VM3.xml. After that, the following command was executed to create the VM taking as input this file: `virsh define /mnt/kvm/xmls/VM3.xml`. The content of the file is included in Appendix C.

Once the VM was created, it was booted from the ISO file downloaded from the Microsoft website to start the installation process. The product key text box was left empty and the option Windows Server 2008 Standard (Full Installation) was selected at the Windows edition selection screen. The type of installation chosen was Custom (Advanced). It installed a clean copy of Windows in the hard drive of the VM, which was detected as Disk 0 and automatically partitioned and formatted by the Windows installer.

When the installation process was completed, the VM was booted into the new installed Windows Server 2008 system. During the first boot, the password for the administrator account was generated:

- User name: Administrator
- Password: 2hLMmVGt

The network interface was configured with the IP address 192.168.10.13 (network mask 255.255.255.0, no default gateway, and no DNS servers) and remote desktop access

was allowed only from computers with network level authentication. The firewall was modified to allow remote desktop access (port TCP 3389) and ping (ICMP echo) request. The DNS server role was also installed and a new forward lookup primary zone named vpsnet.com was created. The firewall was automatically modified to allow incoming DNS queries to the port UDP 53 from the network 192.168.10.0/24. Finally, four A records were manually created inside this zone through the DNS manager: vm1 (with IP address 192.168.10.11), vm2 (192.168.10.12), vm3 (192.168.10.13), and vm4 (192.168.10.14).

In summary, two network services were installed and configured in this VM:

- Remote Desktop (port TCP 3389): Remote Desktop access allowed only for user Administrator.
- DNS (port UDP 53): DNS resolution for zone vpsnet.com allowed for everyone.

3.3.3.4 VM4 - Windows Server 2016 Standard (64-bit)

Virtual resources allocated to this VM:

- CPU: 8 cores.
- RAM: 4096MB.
- Hard drive: /dev/lvm-group/VM4_win2016 (50GB).
- Network: virtual interface vnet3 attached to the bridge br0.

The previous resources and other information required to create the VM in the virtualization node were included in the file /mnt/kvm/xmls/VM4.xml. After that, the following command was executed to create the VM taking as input this file: `virsh define /mnt/kvm/xmls/VM4.xml`. The content of the file is included in Appendix C.

Once the VM was created, it was booted from the ISO file downloaded from the Microsoft website to start the installation process. The option Microsoft Server 2016 Standard (Desktop Experience) was selected at the operating system selection screen. The type of installation chosen was Custom: Install Windows only (advanced). It installed a

clean copy of Windows in the hard drive of the VM, which was detected as Drive 0 and automatically partitioned and formatted by the Windows installer.

When the installation process was completed, the VM was booted into the new installed Windows Server 2016 system. During the first boot, the password for the administrator account was generated:

- User name: Administrator
- Password: 2hLMmVGt

The network interface was configured with the IP address 192.168.10.14 (network mask 255.255.255.0, no default gateway, and no DNS servers) and remote desktop access was allowed only from computers with network level authentication. The firewall was modified to allow remote desktop access (port TCP 3389) and ping (ICMP echo) request. The MySQL server (version 8.0.11) was also installed by downloading the 64-bit Microsoft Installer (MSI) package from the MySQL website. This service was configured as a standalone server, to listen for incoming connections on port TCP 3306, and to be automatically started at Windows startup. Only the default root user account was created, with the same password defined for the Windows administrator user account. The Windows firewall was automatically modified by the installer to allow incoming MySQL connections to the port TCP 3306, and the privileges of the MySQL server were manually adjusted to allow root access from the network 192.168.10.0/24.

In summary, two network services were installed and configured in this VM:

- Remote Desktop (port TCP 3389): Remote Desktop access allowed only for user Administrator.
- MySQL (port TCP 3306): MySQL access allowed only for user root.

3.3.3.5 Additional configuration steps

During the OS installation process followed for each VM (or after the OS was installed) the timezone selected was America/Indiana/Indianapolis (or US & Canada Eastern time), the language chosen was English, and the location was United States.

If a VM required Internet access to update or install any additional package or software, its network configuration was temporally modified to access the Internet using the virtualization node as default gateway. The network configuration of the virtualization node was also modified to accomplish this objective. Once the VM was updated and no additional software was needed, the network configuration of the VM and the virtualization node were restored to their previous state.

A customized Volatility profile was compiled on both VMs running GNU/Linux (VM1 and VM2) after they were successfully installed and updated. This profile was required in order to examine the RAM images created with the utility `virsh` as described in section 3.5. Volatility (the framework used in this study to examine the RAM images) includes built-in support for the majority of the Windows systems, but for a GNU/Linux system a customized profile has to be generated (Ligh et al., 2014). This is due to the large number of Linux kernel versions available and to the fact that each kernel can be compiled with a custom configuration (Ligh et al., 2014). The commands executed to generate the profiles for VM1 and VM2 were included in Appendix D.

3.3.4 Laptop Configuration

The laptop was restored to factory settings and all the available updates from Microsoft were applied. The OS installed after the restore and update process was Microsoft Windows 8.1 Pro, version 6.3.9600. Only one user account was generated:

- User account name: `cf1student`
- Password: `2hLMmVGt`
- Account type: Administrator

Besides the OS and the updates applied, the following software was also installed on the laptop to interact with the four VMs as specified in section 3.5.

- Putty SSH client (version 0.70)
- Google Chrome (version 65.0.3325.146)

- FileZilla FTP client (version 3.32.0)
- Command Prompt interface (included in Microsoft Windows)
- Remote Desktop client (included in Microsoft Windows)
- MySQL Workbench (version 6.3.10)

The last step in the laptop configuration process was connecting the USB to Gigabit Ethernet adapter into a USB port. The network interface integrated in the adapter was automatically detected by the operating system and no additional drives were installed. The network interface was listed as Realtek USB GBE Family Controller by Windows device manager. The IP address 192.168.10.20 was assigned to this interface and one Ethernet patch cord was used to connect the laptop to the switch. Finally, the network communication was verified between the laptop, the virtualization node, and the four VMs.

3.4 Testing Conditions

Before starting the execution of the testing procedures, both hard drives of the virtualization node were examined to determine if they had bad sectors. To this purpose, the physical computer was booted from a Ubuntu Desktop 16.04.4 64-bit USB stick and the option Try Ubuntu without installing was selected. Once Ubuntu booted into graphical mode, the keys Ctrl+Alt+F1 were pressed to change from graphical to command line mode. Using this mode, the default user ubuntu (without password) was logged into the system. The command `sudo su` was executed to allow the user ubuntu to execute commands with superuser privileges. Then, the following command was executed: `badblocks -s -v -n /dev/sda`. The utility badblocks searches a device for bad blocks. Option -s was included to show the progress of the process, option -v to report verbose information, option -n to use non-destructive read/write mode, and the device /dev/sda corresponded to the physical hard drive assigned to store the VM hard drives. Once the previous process finished, the following command was executed: `badblocks -s -v -n`

/dev/sdb. The purpose of this command was to perform the same examination on the device /dev/sdb. This device corresponded to the physical hard drive assigned to store the data collected from the VMs in the next stage of the study. Both commands did not report bad sectors.

The testing procedures related to the creation of virtual hard drive and RAM content images were analyzed from an efficiency standpoint. For this reason, the first step was to obtain a reliable and accurate baseline to compare with the results observed during the testing procedures and to determine if they were efficient. The Ubuntu Desktop previously booted from a USB stick was also used to calculate the transfer rates for both physical hard drives and for the RAM. By default, Ubuntu Desktop starts several services which may use hardware resources. The services related to the graphical mode, network functions, and scheduled tasks were stopped to minimize the usage of resources before executing the transfer rate tests. The services stopped were: lightdm, cron, bluetooth, network-manager, cups-browserd, cups, and avahi-daemon. The network interfaces were also disabled with the command `ifconfig down`. The principal objective of stopping these services and disabling the network interfaces was to maximize the availability of resources during the transfer rate tests in order to get an accurate baseline. The next subsections introduce the steps followed to obtain these values, which were fundamental to determine the efficiency of virsh to create virtual hard drive and RAM content images during the next sections of this study.

3.4.1 Hard drives transfer rates

This subsection describes the process followed to calculate read and write transfer rates for both hard drives available in the virtualization node. These values were used then to calculate the data duplication transfer rate from one hard drive to the other and to use it as baseline for efficiency.

3.4.1.1 Read transfer rate for /dev/sda

The command executed to get the read transfer rate for the physical hard drive selected to store the virtual hard drives was: `dd if=/dev/sda4 of=/dev/null bs=512 conv=noerror`. The partition `/dev/sda4` was selected as input device (parameter `if`) because this partition had a size of 339.5GB approximately, which was large enough to get a reliable transfer rate average. The special file `/dev/null` was selected as output device (parameter `of`) because it just discards the data without performing any real writing operation. The parameter `bs=512` indicates a block size of 512 bytes was read at a time. This option was chosen to match the physical sector size of the hard drive and to create a sector-by-sector sequential reading process. The value `noerror` in the parameter `conv` means the process continues after a read error from the input device.

The previous command was executed 10 times. Table 3.4 displays the results of each execution and the average values.

Table 3.4. Read transfer rate for device /dev/sda

Test	Bytes	Seconds	Bytes/sec	MB/sec
1	339,549,880,320	6,317.83	53,744,700.37	53.74
2	339,549,880,320	6,317.98	53,743,424.37	53.74
3	339,549,880,320	6,318.31	53,740,617.40	53.74
4	339,549,880,320	6,318.07	53,742,658.81	53.74
5	339,549,880,320	6,318.43	53,739,596.75	53.74
6	339,549,880,320	6,317.71	53,745,721.21	53.75
7	339,549,880,320	6,318.54	53,738,661.20	53.74
8	339,549,880,320	6,318.40	53,739,851.91	53.74
9	339,549,880,320	6,317.86	53,744,445.16	53.74
10	339,549,880,320	6,318.30	53,740,702.45	53.74
AVG	339,549,880,320	6,318.14	53,742,037.96	53.74

Figure 3.3 illustrates the read transfer rate (in MB/sec) registered. As it can be seen, the values were stable across the tests, which suggested the 53.74MB/s average value was an accurate indicator of the read transfer rate for the device `/dev/sda`.

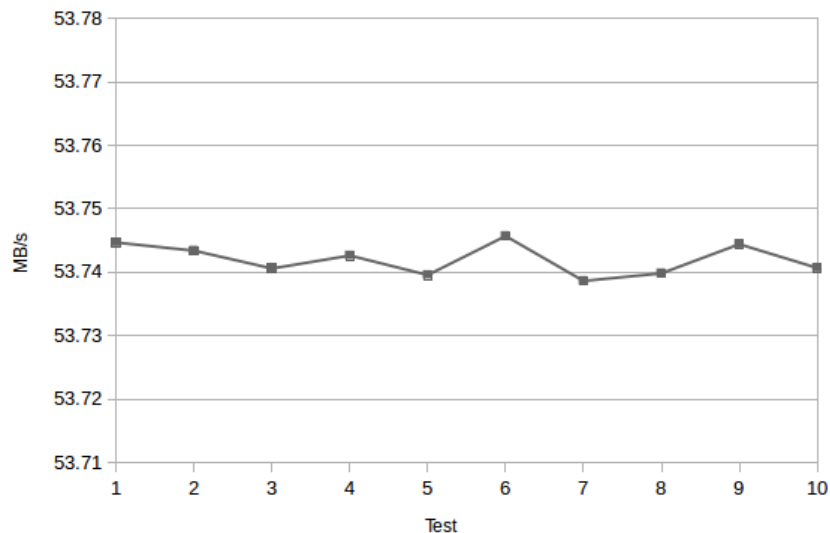


Figure 3.3. /dev/sda read transfer rate

3.4.1.2 Write transfer rate for /dev/sda

The command executed to get the write transfer rate for the physical hard drive selected to store the virtual hard drives was: `dd if=/dev/zero of=/dev/sda4 bs=512 conv=noerror,sync,fdatasync`. The special file `/dev/zero` was selected as input device (parameter `if`) because it provides a stream of null characters and it minimizes the use of CPU compared to other special devices such as `/dev/random` or `/dev/urandom`. The partition `/dev/sda4` was selected as output device (parameter `of`) because this partition was not being used yet and it had a size of 339.5GB approximately, which was large enough to get a reliable transfer rate average. The parameter `bs=512` indicates a block size of 512 bytes was read and written at a time. This option was chosen to match the physical sector size of the hard drive and to create a sector-by-sector sequential writing process. The value `noerror` in the parameter `conv` means the process continues after a read error from the input device. The value `sync` means to pad with null values the destination block in the output device after a read error from the input device. The value `fdatasync` forces each block to be immediately written out to the output device. The principal objective of including this last value was to minimize the impact of the cache and to get a more accurate write transfer rate for the hard drive.

The previous command was executed 10 times. Table 3.5 displays the results of each execution and the average values.

Table 3.5. Write transfer rate for device /dev/sda

Test	Bytes	Seconds	Bytes/sec	MB/sec
1	339,549,880,320	18,111.10	18,748,164.40	18.75
2	339,549,880,320	18,154.20	18,703,654.27	18.70
3	339,549,880,320	18,145.40	18,712,725.01	18.71
4	339,549,880,320	18,057.40	18,803,918.63	18.80
5	339,549,880,320	18,165.70	18,691,813.71	18.69
6	339,549,880,320	18,079.40	18,781,037.00	18.78
7	339,549,880,320	18,152.80	18,705,096.75	18.71
8	339,549,880,320	17,992.70	18,871,535.70	18.87
9	339,549,880,320	17,939.00	18,928,027.22	18.93
10	339,549,880,320	17,941.70	18,925,178.79	18.93
AVG	339,549,880,320	18,073.90	18,787,115.15	18.79

Figure 3.4 illustrates the write transfer rate (in MB/sec) registered. As it can be seen, the values were also stable across the tests, which suggested the 18.79MB/s average value was an accurate indicator of the write transfer rate for the device /dev/sda.

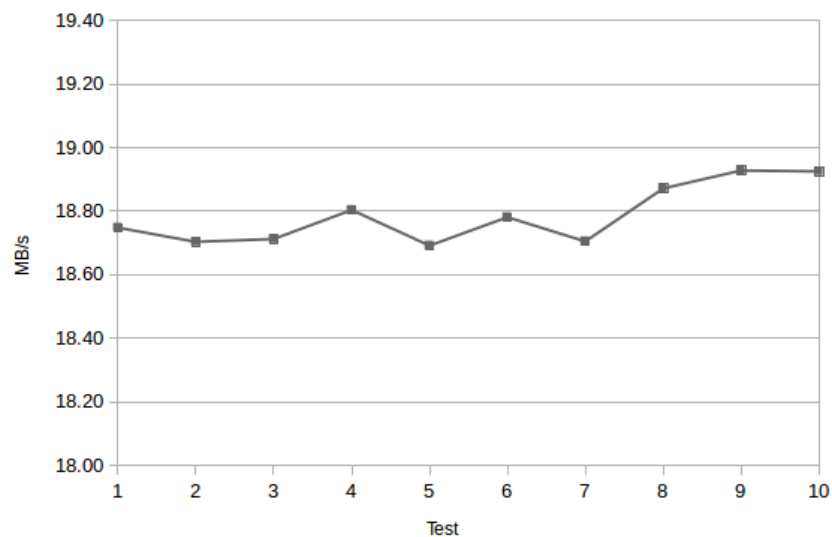


Figure 3.4. /dev/sda write transfer rate

3.4.1.3 Read transfer rate for /dev/sdb

The command executed to get the read transfer rate for the physical hard drive selected to store the information collected from the VMs was: `dd if=/dev/sdb4 of=/dev/null bs=512 conv=noerror`. The purpose of this command was to perform the same previous read transfer rate test on the device /dev/sdb. It was executed 10 times and Table 3.6 displays the results of each execution and the average values.

Table 3.6. Read transfer rate for device /dev/sdb

Test	Bytes	Seconds	Bytes/sec	MB/sec
1	339,549,880,320	2,969.36	114,351,200.37	114.35
2	339,549,880,320	2,969.28	114,354,281.28	114.35
3	339,549,880,320	2,969.38	114,350,430.16	114.35
4	339,549,880,320	2,969.16	114,358,902.96	114.36
5	339,549,880,320	2,969.44	114,348,119.62	114.35
6	339,549,880,320	2,969.40	114,349,659.97	114.35
7	339,549,880,320	2,969.24	114,355,821.80	114.36
8	339,549,880,320	2,969.31	114,353,125.92	114.35
9	339,549,880,320	2,969.38	114,350,430.16	114.35
10	339,549,880,320	2,969.58	114,342,728.71	114.34
AVG	339,549,880,320	2,969.35	114,351,470.10	114.35

Figure 3.5 illustrates the read transfer rate (in MB/sec) registered. As it can be seen, the values were stable across the tests, which suggested the 114.35MB/s average value was an accurate indicator of the read transfer rate for the device /dev/sdb.

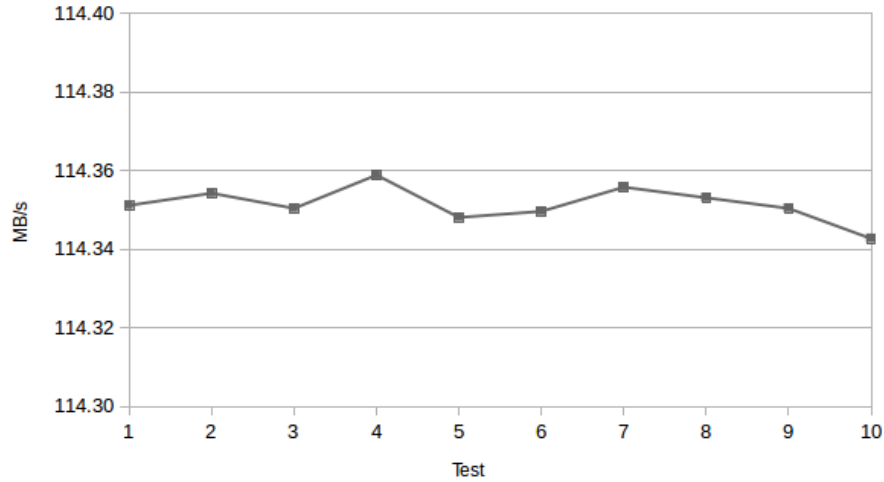


Figure 3.5. /dev/sdb read transfer rate

3.4.1.4 Write transfer rate for /dev/sdb

The command executed to get the write transfer rate for the physical hard drive selected to store the information collected from the VMs was: `dd if=/dev/zero of=/dev/sdb4 bs=512 conv=noerror,sync,fdatasync`. The purpose of this command was to perform the same write transfer rate tests on the device /dev/sdb. It was executed 10 times and Table 3.7 displays the results of each execution and the average values.

Table 3.7. Write transfer rate for device /dev/sdb

Test	Bytes	Seconds	Bytes/sec	MB/sec
1	339,549,880,320	11,127.90	30,513,383.51	30.51
2	339,549,880,320	11,128.10	30,512,835.10	30.51
3	339,549,880,320	11,128.00	30,513,109.30	30.51
4	339,549,880,320	11,129.30	30,509,545.10	30.51
5	339,549,880,320	11,127.60	30,514,206.15	30.51
6	339,549,880,320	11,127.60	30,514,206.15	30.51
7	339,549,880,320	11,128.50	30,511,738.36	30.51
8	339,549,880,320	11,128.30	30,512,286.72	30.51
9	339,549,880,320	11,128.40	30,512,012.54	30.51
10	339,549,880,320	11,128.60	30,511,464.18	30.51
AVG	339,549,880,320	11,128.23	30,512,478.71	30.51

Figure 3.6 illustrates the write transfer rate (in MB/sec) registered. As it can be seen, the values were also stable across the tests, which suggested the 30.51MB/s average value was an accurate indicator of the write transfer rate for the device `/dev/sdb`.

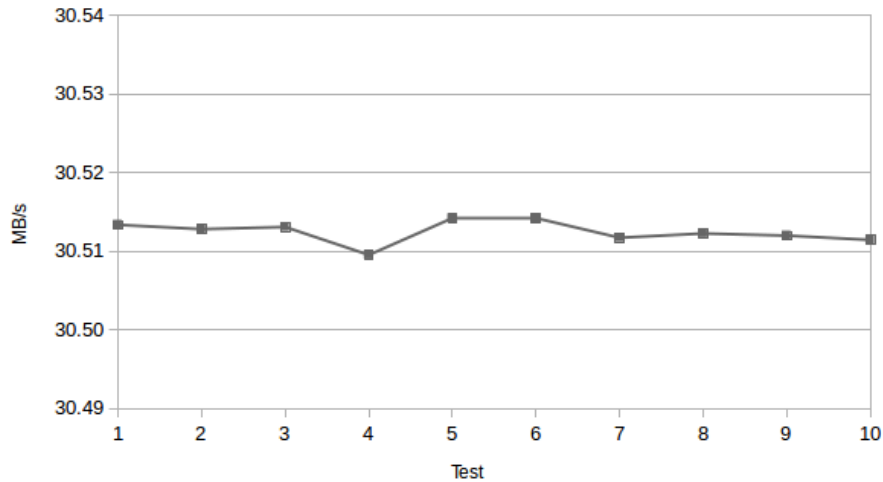


Figure 3.6. /dev/sdb write transfer rate

3.4.1.5 Data duplication transfer rate from `/dev/sda` to `/dev/sdb`

At this point, read and write transfer rate values for both hard drives were known. The final step to get an accurate baseline was testing the process of duplicating data from the hard drive `/dev/sda` (selected to store the VM hard drives) to the hard drive `/dev/sdb` (selected to store the data collected from the VMs). The command executed to get this transfer rate was: `dd if=/dev/sda4 of=/dev/sdb4 bs=512 conv=noerror,sync,fdatasync`. It was executed 10 times and Table 3.8 displays the results of each execution and the average values.

Table 3.8. Data duplication transfer rate from /dev/sda to /dev/sdb

Test	Bytes	Seconds	Bytes/sec	MB/sec
1	339,549,880,320	11,321.50	29,991,598.31	29.99
2	339,549,880,320	11,323.60	29,986,036.27	29.99
3	339,549,880,320	11,322.20	29,989,744.07	29.99
4	339,549,880,320	11,322.70	29,988,419.75	29.99
5	339,549,880,320	11,324.30	29,984,182.72	29.98
6	339,549,880,320	11,322.80	29,988,154.90	29.99
7	339,549,880,320	11,325.70	29,980,476.29	29.98
8	339,549,880,320	11,321.40	29,991,863.23	29.99
9	339,549,880,320	11,317.30	30,002,728.59	30.00
10	339,549,880,320	11,325.50	29,981,005.72	29.99
AVG	339,549,880,320	11,322.70	29,988,420.99	29.99

Figure 3.7 illustrates the data duplication transfer rate (in MB/sec) registered. As it can be seen, the values were stable across the tests, which suggested the 29.99MB/s average value was an accurate indicator of the data duplication transfer rate from the device /dev/sda to the device /dev/sdb.

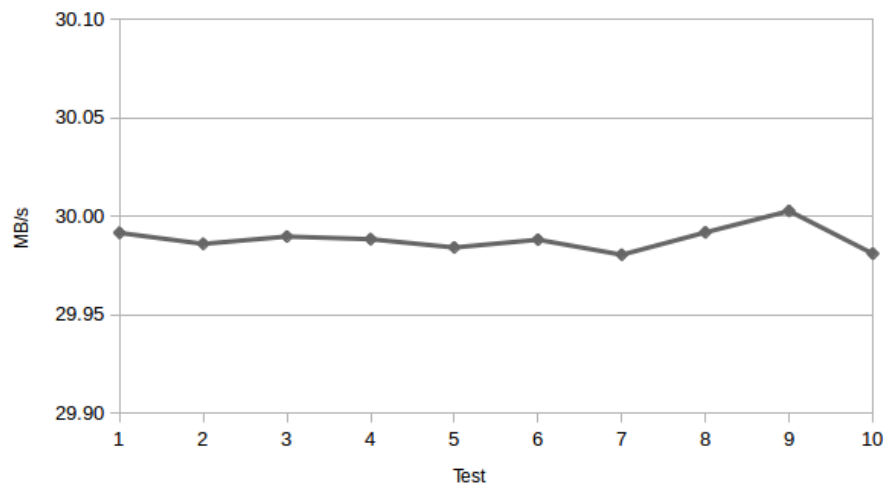


Figure 3.7. Data duplication transfer rate from /dev/sda to /dev/sdb

As it was previously determined, the read transfer rate for /dev/sda was 53.74MB/s and the write transfer rate for /dev/sdb was 30.51MB/s. The motherboard of the

virtualization node had an integrated SATA 3.0 Gbit/s controller, which supported a bandwidth throughput of up to 300MB/s shared between four SATA ports. Of these ports only two were used (one per each hard drive connected), which means if the transfer rate for both hard drives were added, the resulting 84.25MB/s was still much lower than the maximum bandwidth throughput supported by the controller. Therefore, the process to measure the data duplication transfer rate was not limited by the SATA controller, it was limited only by the write transfer rate for the hard drive `/dev/sdb`. This analysis is confirmed by the previous result, since the average data duplication transfer rate (29.99MB/s) was lower, but indeed close, to the write transfer rate for `/dev/sdb` (30.51MB/s).

3.4.2 RAM data duplication transfer rate

The Ubuntu Desktop previously booted from a USB stick was also used to create two RAM disks, to calculate the data duplication transfer rate between them, and to use this value as baseline for efficiency. This approach was employed because in section 3.5 the RAM allocated to the VMs was imaged into a file stored in a RAM disk, which is just a block of RAM that simulates a traditional hard drive. If the RAM content were imaged to a file stored in a hard drive, the process would be limited by the hard drive write transfer rate and it would not represent an accurate value to analyze the efficiency of the imaging process. For this reason, the usage of RAM disks to calculate the RAM data duplication transfer rate seemed to be more appropriate.

To accomplish this objective, two new directories were created by executing the command: `mkdir /mnt/ramdisk{1,2}`. After that, two RAM disks were defined and mounted on these directories by executing: `mount -t tmpfs -o size=12288M tmpfs /mnt/ramdisk{1,2}`. The parameter `size` specified an extension of 12GiB for each RAM disk. A new file called `memory.img` was generated to fill all the space available on the first RAM disk by executing: `dd if=/dev/urandom of=/mnt/ramdisk1/memory.img`. The device `/dev/urandom` was selected as input device (parameter `if`) because it is a special file that provides a stream of random numbers. The previous command wrote 25,165,825 sectors

of 512 bytes to generate a file with a total size of 12,884,902,400 bytes. The file previous generated was then copied into the second RAM disk, by executing the command: `time cp /mnt/ramdisk1/memory.img /mnt/ramdisk2/memory.img`. The program time was included in the command to report the real time needed by the process to complete the copy. The command was executed 10 times. Table 3.9 displays the results of each execution and the average values.

Table 3.9. RAM data duplication transfer rate

Test	Seconds	Bytes/sec	MB/sec
1	9.853	1,307,713,630.37	1,307.71
2	9.860	1,306,785,233.27	1,306.79
3	9.842	1,309,175,208.29	1,309.18
4	9.846	1,308,643,347.55	1,308.64
5	9.841	1,309,308,241.03	1,309.31
6	9.856	1,307,315,584.42	1,307.32
7	9.837	1,309,840,642.47	1,309.84
8	9.842	1,309,175,208.29	1,309.18
9	9.851	1,307,979,129.02	1,307.98
10	9.842	1,309,175,208.29	1,309.18
AVG	9.847	1,308,511,091.30	1,308.51

Figure 3.8 illustrates the data duplication transfer rate (in MB/sec) registered between both RAM disks. As it can be seen, the values were stable across the tests, which suggested the 1,308.51MB/s average value was an accurate indicator of the RAM data duplication transfer rate.

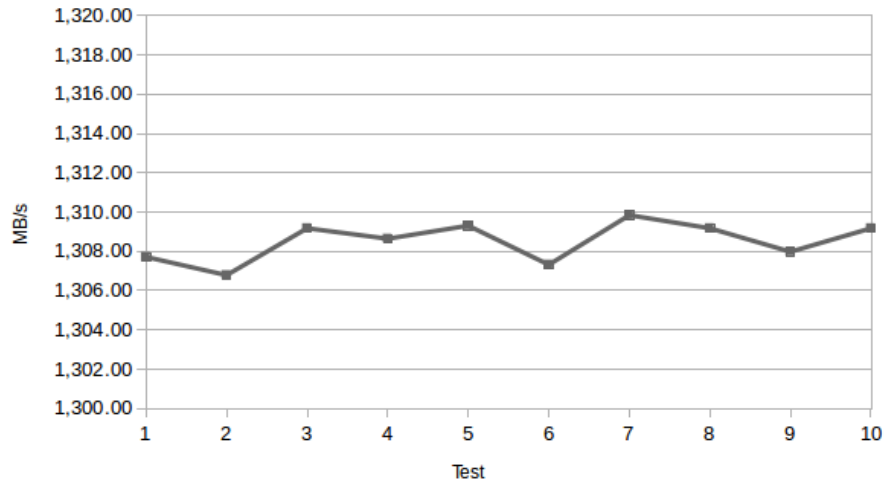


Figure 3.8. RAM data duplication transfer rate

Once this process was concluded, the virtualization node was powered off, the Ubuntu Desktop USB stick was disconnected, and the Ubuntu Server previously installed was booted again.

3.5 Testing Procedures

This section describes the testing procedures performed after the virtualization node, the VMs, and the laptop were implemented (according to section 3.3) and the transfer rates determined (section 3.4). These procedures employed the utilities `virsh` and `tcpdump` to collect digital data from the VMs and to test the hypotheses previously defined (section 3.1).

3.5.1 Virtual hard drive image creation

The testing procedures related to the creation of virtual hard drive images involved an automatic component completed by a BASH script, and a manual component completed by examining the output generated by the script.

The BASH script was developed to test the imaging process of virtual hard drives by employing the utility `virsh` to create a snapshot. This script was executed once per day, during 14 consecutive days, in order to create two images from the hard drive of each VM. One image was stored into a local file in the partition `/dev/sdb4`, which was previously formatted with `ext4` and mounted on the directory `/mnt/forensic-data`. The second image was written into the physical partition `/dev/sdb2`, which was created with the same size of the four virtual hard drives. The main reason for creating two images was to examine if different transfer rates were observed when the image was written directly to physical sectors of a partition and when it was written to a file in a filesystem. The source code of the script was included in Appendix E and the steps completed by it are detailed next:

1. The utility `virsh` was used to create an external snapshot of the hard drive of the first VM. This means the virtual hard drive was set as read-only and a new overlay file was created to record the writing operations on it. This step was completed without suspending the VM and the time needed to create the snapshot was registered.
2. A bit-stream image was created through the command `dd` from the read-only hard drive into the physical partition `/dev/sdb2`. The size of this partition matched the one of the virtual hard drive. The transfer rate of this operation was registered.
3. A second bit-stream image was created through the command `dd` from the read-only hard drive into a local file inside the directory `/mnt/forensic-data`. The transfer rate of this operation was registered.
4. MD5 and SHA1 hash values were calculated for the read-only virtual hard drive, for the physical partition `/dev/sdb2`, and for the file generated in the directory `/mnt/forensic-data`. The objective of this step was to verify if the hash values matched.
5. The principal partition where the VM OS had been installed was mounted on the local directory `/mnt/temp`, using one of the images previously created. A log file located inside this mounted partition was checked to verify if the date and time of the last entry recorded was close to the moment when the snapshot was created.

6. The overlay file was merged into the original virtual hard drive to revert the changes made in the first step. The virtual hard drive was set again as a read-write device. This step was completed without suspending the VM and the time needed to merge the snapshot was registered. Finally, the overlay file was removed.
7. The process started again from the first step, but focusing on the next VM.

Once the script completed the previous steps for the four VMs, its output was manually analyzed to determine the following:

1. If the MD5 and SHA1 hash values calculated for the two images and the virtual hard drive matched.
2. If it was possible to mount, from one of the images created, the principal partition where the VM OS had been installed.
3. If the content of a log file located in the partition mounted reported a close date and time to the moment the snapshot was created.

3.5.2 RAM image creation and network traffic capturing

The testing procedures related to RAM content imaging and real-time network traffic capturing required the cooperation of five participants in order to interact with the four VMs. To this purpose, the participants used the laptop that was previously connected to the emulated cloud provider's network and they interacted with the VMs through different network services. Figure 3.9 illustrates the network connections established from the laptop by the participants and the network services running on each VM.

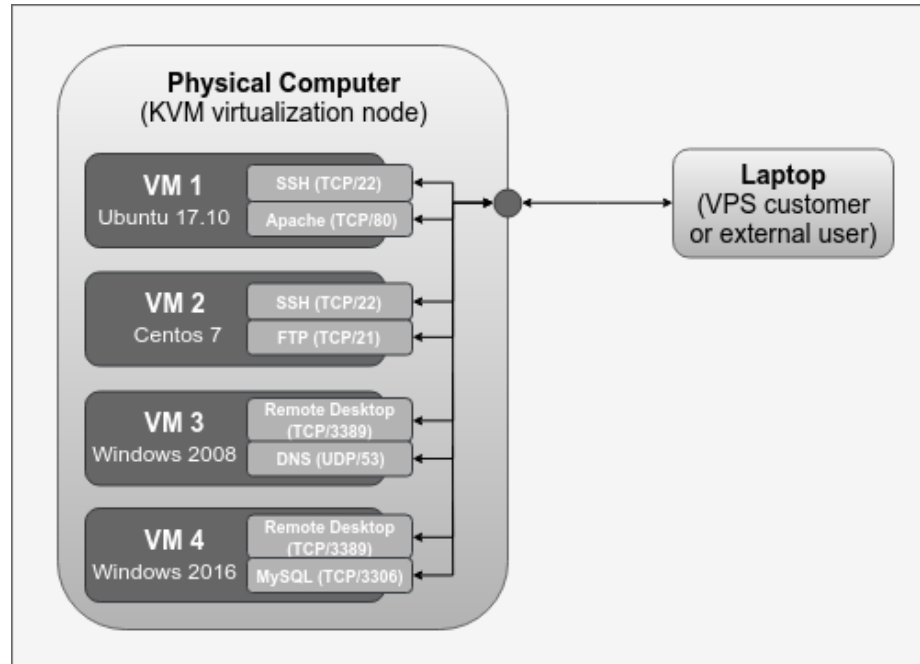


Figure 3.9. Network connections established from the laptop by the participants

The participants followed sequentially four different scripts (one at a time) that detailed the steps to be executed for each VM. Table 3.10 summarizes these steps.

Table 3.10. Steps followed by the participants to interact with the VMs

Step	VM	Description
1	ALL	Log in the laptop.
2	ALL	Send a random number of ICMP echo requests to the VM.
3	VM1	Use the SSH server on VM1 to execute two commands.
	VM2	Use the SSH server on VM2 to execute two commands.
	VM3	Use the remote desktop server on VM3 to execute two commands.
	VM4	Use the remote desktop server on VM4 to execute two commands.
4	VM1	Use a web browser to access the web server on VM1.
	VM2	Use a FTP client to access the FTP server on VM2.
	VM3	Use a DNS client to query the DNS server on VM3.
	VM4	Use a MySQL client to access the MySQL server on VM4.
5	ALL	Complete a table indicating the date and time the interaction started, the random number of ICMP echo requests specified at step 2, and the two commands executed at step 3.

The four complete scripts and the results registered by the participants are included in Appendices G, H, I, and J. Appendix F includes the review exemption letter provided by Purdue's Institutional Review Board (IRB) for this research.

Each participant completed the process of interacting sequentially with the four VMs twice, but at different dates. In addition, only one participant completed the process per each day. The purpose of this segmentation was to allow the researcher to be present during each interaction in case the participants had any concerns about the process to follow and also to simplify the later analysis of the data collected.

Before each participant started the interaction with a particular VM, the utility `tcpdump` was executed on the virtualization node to capture continuously all the network traffic that traversed the TAP interface of the VM. For example, the following command was employed to capture all the network traffic of VM1 (whose TAP interface was `vnet0`) and to store the data collected into a local file: `tcpdump -nn -s0 --interface=vnet0 -w /mnt/forensic-data/net-captures/vm1-test1.pcap`. The meaning of the parameters included in the command were explained in 2.2.2.6.

During the interaction, the participants generated network traffic from the laptop, as specified in table 3.10. Once the interaction was completed, the `tcpdump` process was stopped and the resulting file was examined through the application Wireshark (version 2.2.6). The objective of this examination was to retrieve:

1. The random number of ICMP echo requests sent from the laptop to the VM.
2. The SSH or remote desktop access established from the laptop to the VM.
3. The network service connection (HTTP, FTP, MySQL, or DNS) established from the laptop to the VM.

Similarly, once the interaction with a VM was completed, its RAM content was imaged into a file stored in a RAM disk that was mounted on the directory `/mnt/ramdisk`. The RAM disk was created by executing the command: `mount -t tmpfs -o size=5000M tmpfs /mnt/ramdisk`. The parameter `size` specified an extension of 5GB for the RAM disk. This size was chosen in order to have enough space to store one RAM image created from

any VM (all the VMs had 4GB RAM assigned). The principal and only purpose of using a RAM disk to store the RAM image file was to analyze the efficiency of the imaging process.

The utility `virsh` was executed to create a RAM image from each VM after each participant finished the interaction. For instance, the following command was employed to create a RAM image of VM1 into the RAM disk, and also to report the time needed to finish the process: `time virsh dump VM1_Ubuntu_17.10 /mnt/ramdisk/vm1-ram.dump --memory-only`. The meaning of the parameters included in the command were explained in 2.2.2.4.

Once the previous command ended, the resulting file was moved to the directory `/mnt/forensic-data/ram-images` to be examined through the Volatility Framework (version 2.6). The objective of this examination was to retrieve:

1. The SSH or remote desktop access established from the laptop to the VM, that was intentionally left active during the interaction.
2. The commands executed on the VM OS through the SSH or remote desktop session.
3. The network service connection (HTTP, FTP, MySQL, or DNS) established from the laptop to the VM.

3.6 Measurements for Evaluation

This section describes the procedures followed to determine if each test described in the previous section achieved, after being executed, the expected results for the variables efficiency, integrity, and completeness.

3.6.1 Hypothesis One

The results reported by the BASH script developed to test the creation of virtual hard drive images were examined to verify if the process was efficient and if it respected

the integrity and completeness of the data acquired. Table 3.11 summarizes the steps executed by the BASH script for each image created and the expected result for each step.

Table 3.11. Steps and expected results of the virtual hard drive image creation process

Step	Description	Result
1	Create an external snapshot of the virtual hard drive. This means the virtual hard drive was set as read-only device and a new overlay file was created.	Time
2	Create a bit-stream image from the read-only hard drive into the physical partition /dev/sdb2.	Time
3	Create a bit-stream image from the read-only hard drive into a local file inside the directory /mnt/forensic-data.	Time
4	Calculate MD5 and SHA1 hash values for the read-only hard drive, the physical partition /dev/sdb2, and the file inside /mnt/forensic-data. Did they match?	Boolean
5	Mount the principal partition where the VM OS was installed using one of the images previously created. Was the partition successfully mounted?	Boolean
6	Check date and time of the last entry recorded in a log file located in the partition mounted in the previous step. Were date and time close (less than 6 hours) to the moment when the snapshot was created?	Boolean
7	Merge the overlay file into the original virtual hard drive to revert the changes made in the first step.	Time

The time results observed for steps 1, 2, and 7 were added together. This sum represented the total amount of time needed to take a new snapshot from the virtual hard drive, to use the snapshot to create a bit-stream image into a physical partition, and to revert the changes made when the snapshot was taken. The total size of the virtual hard drive was then divided by the total amount of time in order to get the transfer rate for the whole operation of creating a new image from a snapshot. The ratio between this value and 29.99MB/s (data duplication transfer rate from /dev/sda to /dev/sdb obtained in

subsection 3.4.1) was calculated. As equation 3.1 states, if the ratio was greater than or equal to 0.7, the process was considered efficient:

$$Efficiency = 1, \text{ if } \frac{\text{transfer rate for snapshot and image creation}}{\text{transfer rate for data duplication (Testing Conditions)}} \geq 0.7 \quad (3.1)$$

The 0.7 threshold value was deliberately selected because a reduction of efficiency in 0.3 could be acceptable from a digital forensic perspective, as long as the process respected the integrity and completeness of the data acquired. For the purpose of this study, these two variables were more significant than efficiency to ensure the acquisition process was reliable and accurate. Furthermore, a diminution of efficiency was expected because the images were created from a physical hard drive of the virtualization node, which was continuously running an OS and the four VMs. The time needed to create the snapshot and then to merge it again into the virtual hard drive was also included when efficiency was measured. For these reasons, a result in the previous equation of 0.7 or higher signified that efficiency was fulfilled.

Similarly, the process of creating two images from the snapshot was considered to accomplish integrity and completeness if the boolean results observed for steps 4, 5, and 6 indicated a positive (true) outcome. In other words, step 4 demonstrated both bit-stream images were exact and authentic copies of the virtual hard drive in view of the fact that their hash values matched. While steps 5 and 6 suggested the images were also successfully created because the partition table was recognized, the main partition was mounted, and the entries of the log file extracted were close enough to the moment the snapshot was taken. As Equations 3.2 and 3.3 state, if these three steps had a positive (true) outcome, integrity and completeness were considered satisfied:

$$Integrity = 1, \text{ if } Step\ 4 = true \ \&\& \ Step\ 5 = true \ \&\& \ Step\ 6 = true \quad (3.2)$$

$$Completeness = 1, \text{ if } Step\ 4 = true \ \&\& \ Step\ 5 = true \ \&\& \ Step\ 6 = true \quad (3.3)$$

3.6.2 Hypothesis Two

In a similar way, the RAM image created after each participant interacted with a particular VM was examined to verify if the process was efficient and if it respected the integrity and completeness of the data acquired. Table 3.12 summarizes the steps manually executed and the expected result for each step.

Table 3.12. Steps and expected results of the RAM image creation process

Step	Description	Result
1	Create a RAM content image from the VM (into a RAM disk), once the participant finished the interaction.	Time
2	Use Volatility to recover from the RAM image the SSH or remote desktop connection established by the participant. Was the information successfully retrieved?	Boolean
3	Use Volatility to recover from the RAM image the commands executed on the VM through the SSH or remote desktop session. Was the information successfully retrieved?	Boolean
4	Use Volatility to recover from the RAM image the process name and port used by the network services running on the VM and accessed by the participant. Was the information successfully retrieved?	Boolean

The time result observed for step 1 represented the total amount of time needed to create a RAM content image. The total size of file generated in the RAM disk was then divided by the total amount of time in order to get the transfer rate for the operation of creating a new RAM content image. The ratio between this value and 1,308.51MB/s (RAM data duplication transfer rate obtained in subsection 3.4.2) was calculated. As equation 3.4 states, if the ratio was greater than or equal to 0.5, the process was considered efficient:

$$Efficiency = 1, \text{ if } \frac{\text{transfer rate for RAM content image creation}}{\text{transfer rate for RAM data duplication (Testing Conditions)}} \geq 0.5 \quad (3.4)$$

The 0.5 threshold value was also deliberately selected because a reduction of efficiency in 0.5 could be acceptable from a digital forensic perspective, as long as the

process respected the integrity and completeness of the data acquired. This value represented a more flexible threshold than the one used to analyze efficiency during the hard drive imaging process. There were two main reasons to explain why this threshold was set to a more flexible value: in the first place RAM transfer rates are sensible and more difficult to accurately measure than hard drive transfer rates, and in the second place RAM transfer rate is considerably faster than hard drive transfer rates. For these particularities, a result in the previous equation of 0.5 or higher signified that efficiency was fulfilled.

Similarly, the process of creating a RAM content image was considered to accomplish integrity and completeness if the boolean results observed for steps 2, 3, and 4 indicated a positive (true) outcome. In other words, it was possible to recover from the RAM image the SSH or remote desktop connection established to the VM, the commands executed through this connection, and the information regarding network services running on the VM. As Equations 3.5 and 3.6 state, if these three steps had a positive (true) outcome, integrity and completeness were considered satisfied:

$$Integrity = 1, \text{ if } Step\ 2 = true \ \&\& \ Step\ 3 = true \ \&\& \ Step\ 4 = true \quad (3.5)$$

$$Completeness = 1, \text{ if } Step\ 2 = true \ \&\& \ Step\ 3 = true \ \&\& \ Step\ 4 = true \quad (3.6)$$

3.6.3 Hypothesis Three

The network traffic captured after each interaction with a particular VM was analyzed to verify if the process respected the integrity and completeness of the data acquired. Table 3.13 summarizes the steps manually executed to examine the network traffic file and the expected result for each step.

Table 3.13. Steps and expected results of the network traffic capturing process

Step	Description	Result
1	Use Wireshark to recover the random number of ICMP echo requests sent by the participant. Was the information successfully retrieved?	Boolean
2	Use Wireshark to recover the SSH or remote desktop connection established by the participant. Was the information successfully retrieved? Did it match the information retrieved from the RAM image at step 3 in the previous subsection?	Boolean
3	Use Wireshark to recover the network service connection established by the participant. Was the information successfully retrieved? Did it match the information retrieved from the RAM image at step 4 in the previous subsection?	Boolean

The process of capturing network traffic in real-time was considered to accomplish integrity and completeness if the boolean results observed for steps 1, 2, and 3 indicated a positive (true) outcome. In other words, it was possible to recover from the network traffic file the random number of ICMP echo requests sent to the VM, the SSH or remote desktop connection established to the VM, and the network connection established to the services running on the VM. In addition, the information recovered matched the one observed after examining the results of the RAM imaging process of the same interaction. As Equations 3.7 and 3.8 state, if these three steps had a positive (true) outcome, integrity and completeness were considered satisfied:

$$Integrity = 1, \text{ if } Step\ 1 = true \ \&\& \ Step\ 2 = true \ \&\& \ Step\ 3 = true \quad (3.7)$$

$$Completeness = 1, \text{ if } Step\ 1 = true \ \&\& \ Step\ 2 = true \ \&\& \ Step\ 3 = true \quad (3.8)$$

3.7 Measurements for Success

This section describes the procedures followed to accept or reject each of the three hypotheses stated, based on the measures observed for the variables efficiency, integrity and completeness on each of the tests performed.

3.7.1 Hypothesis One

Fourteen virtual hard drive images were created for each of the four VM, to produce 56 different images in total. For every image creation process, the variables efficiency, integrity, and completeness were assigned a value of 1 if the process met the conditions described in section 3.6. Otherwise, the variables were assigned a value of 0. Based on these values, equation 3.9 states the condition to accept H_1 .

$$H_1 : Integrity * (10) + Completeness * (10) + Efficiency * (5) \geq 20 \quad (3.9)$$

The previous equation conferred a weight of 10 to the variables integrity and completeness and a weight of 5 to the variable efficiency. H_1 was accepted if the sum of the three variables by their weight was greater than or equal to 20 for every image creation process completed or otherwise it was rejected. In other words, H_1 was accepted if at least integrity and completeness were achieved for each image created. As it was mentioned before, from a digital forensic perspective a reduction of efficiency could be acceptable as long as the process respects the integrity and completeness of the data acquired. For the purpose of this study, these two variables were more significant than efficiency to ensure the acquisition process was reliable and accurate.

3.7.2 Hypothesis Two

Ten RAM images were created for each of the four VM, to produce 40 different images in total. For every RAM image creation process, the variables efficiency, integrity,

and completeness were assigned a value of 1 if the process met the conditions described in section 3.6. Otherwise, the variables were assigned a value of 0. Based on these values, equation 3.10 states the condition to accept H_2 .

$$H_2 : Integrity * (10) + Completeness * (10) + Efficiency * (5) \geq 20 \quad (3.10)$$

The previous equation conferred a weight of 10 to the variables integrity and completeness and a weight of 5 to the variable efficiency. H_2 was accepted if the sum of the three variables by their weight was greater than or equal to 20 for every RAM image creation process completed or otherwise it was rejected. In other words, H_2 was accepted if at least integrity and completeness were achieved for each RAM image created. As it was mentioned before, from a digital forensic perspective a reduction of efficiency could be acceptable as long as the process respects the integrity and completeness of the data acquired. For the purpose of this study, these two variables were more significant than efficiency to ensure the acquisition process was reliable and accurate.

3.7.3 Hypothesis Three

Ten network capturing processes were executed for each of the four VM, to produce 40 different network traffic files. For every network capturing process, the variables integrity and completeness were assigned a value of 1 if the process met the conditions described in section 3.6. Otherwise, the variables were assigned a value of 0. Based on these values, equation 3.11 states the condition to accept H_3 .

$$H_3 : Integrity * (10) + Completeness * (10) \geq 20 \quad (3.11)$$

In this case, the part of the equation related to efficiency was removed because this variable was not considered for the process of capturing network traffic through the utility tcpdump.

3.8 Threats to Validity

The cooperation of participants in the study may introduce uncertainty. For instance, they could omit one step in the procedures detailed by the scripts and the later analysis of the RAM image or network traffic files could get inaccurate results. In order to minimize this threat, the scripts were written as detailed as possible to guide the participants, step by step, throughout the study. The researcher was also present during each interaction in case the participants had any concerns about the process to follow. The objective of these decisions was to minimize the occurrence of unexpected events when the participants interacted with the VMs that could impact on the validity of the results.

Additionally, the participants were not informed about the global scope of the research study. They were only introduced to the scripts to follow during the interaction, with the principal purpose of minimizing the introduction of bias in their behavior.

In terms of reliability, if the study were repeated using the same software versions on the virtualization node, the results observed should be replicated. The following list shows the critical software and corresponding version for the purpose of this study:

- Virtualization node OS: Ubuntu Server 16.04.4 64-bit.
- Linux kernel and KVM module (package linux-image-generic): 4.4.0-116.140.
- KVM user-space tools (package qemu-kvm): 2.5.0.
- Libvirt service (package libvirt0) and Libvirt client (package libvirt-bin): 1.3.1-1.
- Tcpdump: 4.9.2-0.

Appendix A shows a complete list of all the packages installed on the virtualization node, including their respective version.

It is probable that the results were also replicated if different software versions were used, specially the GNU/Linux distribution and the Linux kernel installed on the virtualization node. However, other software versions were not analyzed in this study and more research is needed to support this observation.

A different physical computer could be used as virtualization node as long as the processor includes virtualization technology support and its hardware components are fully supported by the Linux kernel. The rest of the physical devices (Ethernet switch and Laptop) are not relevant as long as they work appropriately and meet the minimum requirements for the purpose of the study.

3.9 Summary

This chapter provided a detailed description of the methodology used in this research study. It presented the research question, hypotheses, participants, research design, testing conditions and procedures, measurements for evaluation and success, and possible threats to validity.

CHAPTER 4. RESULTS

This chapter presents the results registered after completing the testing procedures previously described for each of the hypotheses stated. The chapter also interprets the results to determine whether or not each hypothesis was accepted.

4.1 Hypothesis One

Fourteen different images were created from the virtual hard drive assigned to VM1. This virtual hard drive and each image generated had 97,656,832 sectors of 512 bytes and a total size of 50,000,297,984 bytes. Table 4.1 summarizes the results of the different steps executed to analyze each image created. The column called TTR displays the total transfer rate for each image creation process, including the steps needed to create and merge the external snapshot. This means the values reported by this column take into account the time results of steps 1, 2, and 7, previously described in section 3.6.1. The column named Ratio represents the relationship between column TTR and the data duplication transfer rate used as a baseline (29.99MB/s). The value of the variable efficiency depended on this column: each image creation process was considered efficient if the ratio value was greater than or equal to 0.7. The possible values of columns S4, S5, and S6 were T (true) or F (false) according to the results observed for steps 4, 5, and 6, described in section 3.6.1. The values of the variables integrity and completeness depended on these columns: each image creation process was considered to respect the integrity and completeness of the data acquired if the results reported by these three columns were T (true). In other words, the last four columns (Ratio, S4, S5, and S6) provided the information required to assign a value of 0 or 1 to the variables efficiency, integrity, and completeness for each image creation process completed.

Table 4.1. VM1 hard drive image creation process

Test	SCT	DDT	SMT	DDTR	TTR	Ratio	S4	S5	S6
1	1.77s	1,977.11s	4.20s	25.29MB/s	25.21MB/s	0.84	T	T	T
2	1.86s	2,010.38s	3.37s	24.87MB/s	24.81MB/s	0.83	T	T	T
3	1.69s	1,984.36s	3.42s	25.20MB/s	25.13MB/s	0.84	T	T	T
4	1.79s	1,945.54s	3.54s	25.70MB/s	25.63MB/s	0.85	T	T	T
5	1.78s	1,979.25s	3.45s	25.26MB/s	25.20MB/s	0.84	T	T	T
6	1.69s	1,955.87s	3.25s	25.56MB/s	25.50MB/s	0.85	T	T	T
7	1.77s	2,060.47s	3.35s	24.27MB/s	24.21MB/s	0.81	T	T	T
8	1.80s	1,966.50s	3.39s	25.43MB/s	25.36MB/s	0.85	T	T	T
9	1.77s	1,961.44s	6.75s	25.49MB/s	25.38MB/s	0.85	T	T	T
10	1.82s	1,925.99s	3.30s	25.96MB/s	25.89MB/s	0.86	T	T	T
11	1.73s	1,955.68s	3.67s	25.57MB/s	25.50MB/s	0.85	T	T	T
12	1.70s	1,962.14s	3.72s	25.48MB/s	25.41MB/s	0.85	T	T	T
13	1.64s	1,931.43s	3.17s	25.89MB/s	25.82MB/s	0.86	T	T	T
14	1.79s	2,056.23s	3.38s	24.32MB/s	24.26MB/s	0.81	T	T	T
AVG	1.76s	1,976.60s	3.71s	25.31MB/s	25.24MB/s	0.84	-	-	-

Note. SCT represents the external snapshot creation time (step 1), DDT the bit-stream image creation time (step 2), and SMT the external snapshot merging time (step 7). DDTR represents the transfer rate recorded only for the bit-stream image creation process, while TTR the total transfer rate, which includes the time needed to create and merge the external snapshot in addition to the bit-stream image creation time. Ratio represents the relationship between TTR and the value used as a baseline (29.99MB/s). S4, S5, and S6 represent the results of steps 4, 5, and 6, respectively. The possible results of S4, S5, and S6 were T (true) or F (false).

Fourteen different images were also created from the virtual hard drive assigned to VM2. This virtual hard drive and each image generated had the same size previously detailed for VM1. Table 4.2 summarizes the results of the different steps executed to analyze each image created.

Table 4.2. VM2 hard drive image creation process

Test	SCT	DDT	SMT	DDTR	TTR	Ratio	S4	S5	S6
1	1.94s	1,845.62s	3.49s	27.09MB/s	27.01MB/s	0.90	T	T	T
2	2.02s	1,917.03s	3.07s	26.08MB/s	26.01MB/s	0.87	T	T	T
3	1.78s	1,898.53s	3.49s	26.34MB/s	26.26MB/s	0.88	T	T	T
4	1.89s	1,884.68s	3.08s	26.53MB/s	26.46MB/s	0.88	T	T	T
5	1.74s	1,864.89s	2.92s	26.81MB/s	26.74MB/s	0.89	T	T	T
6	1.88s	1,870.41s	2.82s	26.73MB/s	26.67MB/s	0.89	T	T	T
7	1.91s	1,905.96s	3.02s	26.23MB/s	26.17MB/s	0.87	T	T	T
8	1.95s	1,860.15s	3.04s	26.88MB/s	26.81MB/s	0.89	T	T	T
9	1.93s	1,845.36s	3.86s	27.10MB/s	27.01MB/s	0.90	T	T	T
10	1.88s	1,864.82s	3.48s	26.81MB/s	26.74MB/s	0.89	T	T	T
11	1.87s	1,866.35s	3.42s	26.79MB/s	26.71MB/s	0.89	T	T	T
12	1.92s	1,911.01s	3.96s	26.16MB/s	26.08MB/s	0.87	T	T	T
13	1.75s	1,799.27s	4.80s	27.79MB/s	27.69MB/s	0.92	T	T	T
14	1.96s	1,914.25s	4.90s	26.12MB/s	26.03MB/s	0.87	T	T	T
AVG	1.89s	1,874.88s	3.53s	26.68MB/s	26.60MB/s	0.89	-	-	-

Note. SCT represents the external snapshot creation time (step 1), DDT the bit-stream image creation time (step 2), and SMT the external snapshot merging time (step 7). DDTR represents the transfer rate recorded only for the bit-stream image creation process, while TTR the total transfer rate, which includes the time needed to create and merge the external snapshot in addition to the bit-stream image creation time. Ratio represents the relationship between TTR and the value used as a baseline (29.99MB/s). S4, S5, and S6 represent the results of steps 4, 5, and 6, respectively. The possible results of S4, S5, and S6 were T (true) or F (false).

Fourteen different images were also created from the virtual hard drive assigned to VM3. This virtual hard drive and each image generated had the same size previously detailed for VM1. Table 4.3 summarizes the results of the different steps executed to analyze each image created.

Table 4.3. VM3 hard drive image creation process

Test	SCT	DDT	SMT	DDTR	TTR	Ratio	S4	S5	S6
1	2.60s	1,925.07s	3.86s	25.97MB/s	25.89MB/s	0.86	T	T	T
2	2.14s	1,964.93s	3.89s	25.45MB/s	25.37MB/s	0.85	T	T	T
3	2.16s	2,032.38s	3.62s	24.60MB/s	24.53MB/s	0.82	T	T	T
4	2.31s	1,980.01s	3.31s	25.25MB/s	25.18MB/s	0.84	T	T	T
5	1.84s	2,067.05s	3.47s	24.19MB/s	24.13MB/s	0.80	T	T	T
6	2.19s	2,156.22s	3.29s	23.19MB/s	23.13MB/s	0.77	T	T	T
7	2.18s	2,178.07s	3.20s	22.96MB/s	22.90MB/s	0.76	T	T	T
8	1.74s	2,061.69s	3.47s	24.25MB/s	24.19MB/s	0.81	T	T	T
9	2.15s	1,940.10s	4.22s	25.77MB/s	25.69MB/s	0.86	T	T	T
10	2.03s	1,936.74s	3.33s	25.82MB/s	25.75MB/s	0.86	T	T	T
11	2.13s	1,921.73s	3.82s	26.02MB/s	25.94MB/s	0.86	T	T	T
12	2.32s	1,824.84s	3.31s	27.40MB/s	27.32MB/s	0.91	T	T	T
13	2.05s	1,814.18s	3.85s	27.56MB/s	27.47MB/s	0.92	T	T	T
14	2.43s	2,085.50s	3.78s	23.98MB/s	23.90MB/s	0.80	T	T	T
AVG	2.16s	1,992.04s	3.60s	25.17MB/s	25.10MB/s	0.84	-	-	-

Note. SCT represents the external snapshot creation time (step 1), DDT the bit-stream image creation time (step 2), and SMT the external snapshot merging time (step 7). DDTR represents the transfer rate recorded only for the bit-stream image creation process, while TTR the total transfer rate, which includes the time needed to create and merge the external snapshot in addition to the bit-stream image creation time. Ratio represents the relationship between TTR and the value used as a baseline (29.99MB/s). S4, S5, and S6 represent the results of steps 4, 5, and 6, respectively. The possible results of S4, S5, and S6 were T (true) or F (false).

Fourteen different images were also created from the virtual hard drive assigned to VM4. This virtual hard drive and each image generated had the same size previously detailed for VM1. Table 4.4 summarizes the results of the different steps executed to analyze each image created.

Table 4.4. VM4 hard drive image creation process

Test	SCT	DDT	SMT	DDTR	TTR	Ratio	S4	S5	S6
1	1.55s	1,943.80s	7.26s	25.72MB/s	25.61MB/s	0.85	T	T	T
2	1.93s	1,949.18s	5.77s	25.65MB/s	25.55MB/s	0.85	T	T	T
3	2.30s	2,032.45s	7.77s	24.60MB/s	24.48MB/s	0.82	T	T	T
4	2.10s	2,027.28s	8.72s	24.66MB/s	24.53MB/s	0.82	T	T	T
5	2.61s	2,042.94s	6.06s	24.47MB/s	24.37MB/s	0.81	T	T	T
6	2.49s	2,106.78s	6.22s	23.73MB/s	23.64MB/s	0.79	T	T	T
7	2.45s	2,051.54s	5.54s	24.37MB/s	24.28MB/s	0.81	T	T	T
8	2.26s	2,024.43s	6.45s	24.70MB/s	24.59MB/s	0.82	T	T	T
9	1.99s	1,946.88s	4.72s	25.68MB/s	25.59MB/s	0.85	T	T	T
10	1.77s	2,104.86s	4.66s	23.75MB/s	23.68MB/s	0.79	T	T	T
11	1.75s	2,099.97s	4.44s	23.81MB/s	23.74MB/s	0.79	T	T	T
12	1.83s	1,914.80s	4.82s	26.11MB/s	26.02MB/s	0.87	T	T	T
13	1.64s	2,012.17s	4.91s	24.85MB/s	24.77MB/s	0.83	T	T	T
14	1.69s	1,949.67s	6.14s	25.65MB/s	25.54MB/s	0.85	T	T	T
AVG	2.03s	2,014.77s	5.96s	24.84MB/s	24.74MB/s	0.83	-	-	-

Note. SCT represents the external snapshot creation time (step 1), DDT the bit-stream image creation time (step 2), and SMT the external snapshot merging time (step 7). DDTR represents the transfer rate recorded only for the bit-stream image creation process, while TTR the total transfer rate, which includes the time needed to create and merge the external snapshot in addition to the bit-stream image creation time. Ratio represents the relationship between TTR and the value used as a baseline (29.99MB/s). S4, S5, and S6 represent the results of steps 4, 5, and 6, respectively. The possible results of S4, S5, and S6 were T (true) or F (false).

The values presented in the previous four tables suggested the utility virsh is able to take snapshots of virtual hard drives, from where bit-stream images can be created in an efficient manner. The ratio value for each of the 56 different images generated was never lower than the 0.7 threshold value. The lowest ratio value registered was 0.76 while the average ratio value for all the 56 imaging processes was 0.85. In other words, the average transfer rate achieved by the imaging processes was 25.42MB/s while the data duplication transfer rate from /dev/sda to /dev/sdb used as a baseline was 29.99MB/s. This baseline represented the transfer rate in optimal conditions because it was calculated when the availability of hardware resources on the virtualization node was maximized. Conversely, all the virtual hard drive images were created while the virtualization node was using

resources to execute different OS tasks and running the four VMs. For instance, the logging service of the virtualization node was writing to the physical hard drive `/dev/sda` and the four VMs were reading and writing their virtual hard drives stored in the same physical hard drive. This means the hardware resources were shared between the imaging processes and other tasks in execution at the virtualization node. If all these facts are considered, the average value of 0.85 observed for all the imaging processes seems to be notably efficient.

Similarly, the values in the tables suggested the images generated respected the integrity and completeness of the data acquired. Columns S4, S5, and S6 reported always T (true) results for the different verifications after each of the 56 imaging processes finished. Firstly, the MD5 and SHA1 hash values calculated for the virtual hard drive and for the two bit-stream images matched every time, which means the images were exact and authentic copies of the virtual hard drive (S4). Secondly, the main partition of each image created into the physical partition `/dev/sdb2` was successfully mounted on the virtualization node (S5). This step implies two facts: the partition table of the image created was correctly recognized and the filesystem of the main partition of the image was detected and mounted by the virtualization node. Thirdly, the content of a file located inside the mounted partition was successfully accessed and the last entry recorded in this file was always close (less than 6 hours) to the moment when the snapshot was created (S6). These results evidence that all the images created respected the integrity and completeness of the data acquired.

For the previous reasons, H_1 was accepted, which means virsh is able to take snapshots of the virtual hard drives of a VPS from where bit-stream images can be created in an efficient manner, respecting the integrity and completeness of the data acquired.

4.2 Hypothesis Two

Ten different RAM images were created from VM1 and each of them had a total size of 4,429,395,320 bytes (approximately 4GiB). Table 4.5 summarizes the results of the different steps executed to analyze each RAM image created. The column called RITR

displays the total transfer rate for each RAM image creation process based on the time needed to generate the image and its size. This means the values reported by this column take into account the time results of step 1 previously described in section 3.6.2. The column named Ratio represents the relationship between column RITR and the RAM data duplication transfer rate used as a baseline (1,308.51MB/s). The value of the variable efficiency depended on this column: each RAM image creation process was considered efficient if the ratio value was greater than or equal to 0.5. The possible values of columns S2, S3, and S4 were T (true) or F (false) according to the results observed for steps 2, 3, and 4, also described in section 3.6.2. The values of the variables integrity and completeness depended on these columns: each RAM image creation process was considered to respect the integrity and completeness of the data acquired if the results reported by these three columns were T (true). In other words, the last four columns (Ratio, S2, S3, and S4) provided the information required to assign a value of 0 or 1 to the variables efficiency, integrity, and completeness for each RAM image creation process completed.

Table 4.5. VM1 RAM image creation process

Test	RICT	RITR	Ratio	S2	S3	S4
1	5.378s	823.61MB/s	0.63	T	T	T
2	5.333s	830.56MB/s	0.63	T	T	T
3	5.843s	758.07MB/s	0.58	T	T	T
4	5.501s	805.20MB/s	0.62	T	T	T
5	6.142s	721.16MB/s	0.55	T	T	T
6	5.423s	816.77MB/s	0.62	T	T	T
7	5.666s	781.75MB/s	0.60	T	T	T
8	5.407s	819.20MB/s	0.63	T	T	T
9	5.910s	749.47MB/s	0.57	T	T	T
10	5.718s	774.64MB/s	0.59	T	T	T
AVG	5.632s	788.04MB/s	0.60	-	-	-

Note. RICT represents the RAM image creation time (step 1) and RITR the RAM image creation transfer rate. Ratio represents the relationship between RITR and the value used as a baseline (1,308.51MB/s). S2, S3, and S4 represent the results of steps 2, 3, and 4, respectively. The possible results of S2, S3, and S4 were T (true) or F (false).

Ten different RAM images were also created from VM2 and each of them had a total size of 4,429,331,264 bytes. Table 4.6 summarizes the results of the different steps executed to analyze each RAM image created.

Table 4.6. VM2 RAM image creation process

Test	RICT	RITR	Ratio	S2	S3	S4
1	5.463s	810.79MB/s	0.62	T	T	T
2	5.592s	792.08MB/s	0.61	T	T	T
3	5.625s	787.44MB/s	0.60	T	T	T
4	5.701s	776.94MB/s	0.59	T	T	T
5	5.542s	799.23MB/s	0.61	T	T	T
6	5.670s	781.20MB/s	0.60	T	T	T
7	5.576s	794.36MB/s	0.61	T	T	T
8	5.432s	815.41MB/s	0.62	T	T	T
9	5.592s	792.08MB/s	0.61	T	T	T
10	5.676s	780.36MB/s	0.60	T	T	T
AVG	5.587s	792.99MB/s	0.61	-	-	-

Note. RICT represents the RAM image creation time (step 1) and RITR the RAM image creation transfer rate. Ratio represents the relationship between RITR and the value used as a baseline (1,308.51MB/s). S2, S3, and S4 represent the results of steps 2, 3, and 4, respectively. The possible results of S2, S3, and S4 were T (true) or F (false).

Ten different RAM images were also created from VM3 and each of them had a total size of 4,429,395,320 bytes. Table 4.7 summarizes the results of the different steps executed to analyze each RAM image created.

Table 4.7. VM3 RAM image creation process

Test	RICT	RITR	Ratio	S2	S3	S4
1	5.344s	828.85MB/s	0.63	T	T	T
2	5.859s	756.00MB/s	0.58	T	T	T
3	5.975s	741.32MB/s	0.57	T	T	T
4	5.974s	741.45MB/s	0.57	T	T	T
5	5.530s	800.98MB/s	0.61	T	T	T
6	5.990s	739.46MB/s	0.57	T	T	T
7	5.770s	767.66MB/s	0.59	T	T	T
8	6.077s	728.88MB/s	0.56	T	T	T
9	6.041s	733.22MB/s	0.56	T	T	T
10	5.516s	803.01MB/s	0.61	T	T	T
AVG	5.808s	764.08MB/s	0.58	-	-	-

Note. RICT represents the RAM image creation time (step 1) and RITR the RAM image creation transfer rate. Ratio represents the relationship between RITR and the value used as a baseline (1,308.51MB/s). S2, S3, and S4 represent the results of steps 2, 3, and 4, respectively. The possible results of S2, S3, and S4 were T (true) or F (false).

Ten different RAM images were also created from VM4 and each of them had a total size of 4,429,396,856 bytes. Table 4.8 summarizes the results of the different steps executed to analyze each RAM image created.

Table 4.8. VM4 RAM image creation process

Test	RICT	RITR	Ratio	S2	S3	S4
1	5.900s	750.75MB/s	0.57	T	T	T
2	5.550s	798.09MB/s	0.61	T	T	T
3	5.941s	745.56MB/s	0.57	T	T	T
4	5.943s	745.31MB/s	0.57	T	T	T
5	5.355s	827.15MB/s	0.63	T	T	T
6	5.921s	748.08MB/s	0.57	T	T	T
7	5.711s	775.59MB/s	0.59	T	T	T
8	6.081s	728.40MB/s	0.56	T	T	T
9	5.977s	741.07MB/s	0.57	T	T	T
10	5.145s	860.91MB/s	0.66	T	T	T
AVG	5.752s	772.09MB/s	0.59	-	-	-

Note. RICT represents the RAM image creation time (step 1) and RITR the RAM image creation transfer rate. Ratio represents the relationship between RITR and the value used as a baseline (1,308.51MB/s). S2, S3, and S4 represent the results of steps 2, 3, and 4, respectively. The possible results of S2, S3, and S4 were T (true) or F (false).

The values presented in the previous four tables suggested the utility *virsh* is able to create images of the RAM content in an efficient manner. The ratio value for each of the 40 different RAM images generated was never lower than the 0.5 threshold value. The lowest ratio value registered was 0.55 while the average ratio value for all the 40 RAM imaging processes was 0.6. In other words, the average transfer rate achieved by the imaging processes was 779.30MB/s while the average RAM data duplication transfer rate used as a baseline was 1,308.51MB/s. This baseline represented the transfer rate in optimal conditions because it was calculated when the availability of hardware resources on the virtualization node was maximized. Conversely, all the RAM images were created while the virtualization node was using resources to execute different OS tasks and running the four VMs. This means the hardware resources were shared between the imaging processes and other tasks in execution at the virtualization node. If all these facts are considered, the average value of 0.60 observed for all the RAM imaging processes seems to be efficient.

Similarly, the values in the tables suggested the RAM images generated respected the integrity and completeness of the data acquired. Columns S2, S3, and S4 reported

always T (true) results for the different verifications after each of the 40 RAM images were created. In the first place, it was possible to recover from every RAM image the SSH or remote desktop session left intentionally active by the participants during their interactions (S2). In the second place, the commands executed by the participants through a SSH session (for GNU/Linux VMs) or through a command prompt window (for Windows VMs) were also recovered from every RAM image (S3). In the third place, it was also possible to retrieve the process name and port used by the network services running in each VM and accessed by the participants (S4). These results evidence that all the RAM images created respected the integrity and completeness of the data acquired.

For the previous reasons, H_2 was accepted, which means virsh is able to create images of the RAM content of a VPS in an efficient manner, respecting the integrity and completeness of the data acquired.

4.3 Hypothesis Three

Ten network traffic capture files were generated from VM1, once per each interaction. Table 4.9 summarizes the results of the three different steps executed to analyze each file. The possible values of columns Step 1, Step 2, and Step 3, were T (true) or F (false) according to the results observed for each step, previously described in section 3.6.3. The values of the variables integrity and completeness depended on these results: each network traffic capturing process was considered to respect the integrity and completeness of the data acquired if the results reported by these columns were T (true). In other words, these three columns provided the information required to assign a value of 0 or 1 to the variables integrity and completeness for each network traffic capturing process completed.

Table 4.9. VM1 network traffic capturing process

Test	Step 1	Step 2	Step 3
1	T	T	T
2	T	T	T
3	T	T	T
4	T	T	T
5	T	T	T
6	T	T	T
7	T	T	T
8	T	T	T
9	T	T	T
10	T	T	T

Note. The possible results of Step 1, Step 2, and Step 3 were T (true) or F (false).

Ten network traffic capture files were also generated from VM2. Table 4.10 summarizes the results of the three different steps executed to analyze each file.

Table 4.10. VM2 network traffic capturing process

Test	Step 1	Step 2	Step 3
1	T	T	T
2	T	T	T
3	T	T	T
4	T	T	T
5	T	T	T
6	T	T	T
7	T	T	T
8	T	T	T
9	T	T	T
10	T	T	T

Note. The possible results of Step 1, Step 2, and Step 3 were T (true) or F (false).

Ten network traffic capture files were also generated from VM3. Table 4.11 summarizes the results of the three different steps executed to analyze each file.

Table 4.11. VM3 network traffic capturing process

Test	Step 1	Step 2	Step 3
1	T	T	T
2	T	T	T
3	T	T	T
4	T	T	T
5	T	T	T
6	T	T	T
7	T	T	T
8	T	T	T
9	T	T	T
10	T	T	T

Note. The possible results of Step 1, Step 2, and Step 3 were T (true) or F (false).

Ten network traffic capture files were also generated from VM4. Table 4.12 summarizes the results of the three different steps executed to analyze each file.

Table 4.12. VM4 network traffic capturing process

Test	Step 1	Step 2	Step 3
1	T	T	T
2	T	T	T
3	T	T	T
4	T	T	T
5	T	T	T
6	T	T	T
7	T	T	T
8	T	T	T
9	T	T	T
10	T	T	T

Note. The possible results of Step 1, Step 2, and Step 3 were T (true) or F (false).

The values presented in the previous four tables suggested the utility tcpdump is able to capture in real-time the network traffic of a VPS, respecting the integrity and completeness of the data acquired. Columns Step 1, Step 2, and Step 3 reported always T (true) results for the different verifications after each of the 40 interactions were completed. In the first place, it was possible to recover from every network traffic file the

random number of ICMP echo requests sent by the participants during their interactions (Step 1). In the second place, the details of the SSH or remote desktop session established by the participants were also recovered (Step 2). In the third place, it was also possible to retrieve the details of the network connection established to the services running on each VM and accessed by the participants (Step 3).

In addition, not only the ICMP echo requests were retrieved from the files, but also every single reply associated to each request. The three-way handshake (SYN, SYN-ACK, ACK) defined by the TCP protocol to set up a new connection was also recovered for each SSH and remote desktop session established by the participants. Moreover, the same three-way handshake was retrieved for each TCP connection established to the services running on VM1, VM2, and VM4 (HTTP, FTP, and MySQL, respectively). Similarly, once each of these connections was closed, the termination four-way handshake (FIN, ACK, FIN, ACK) defined by the TCP protocol was also recovered. Even more specific details were found: all the FTP commands sent to and received from the FTP server on VM2 were recovered, including user names, passwords, and content of the files transferred. It was also possible to determine when the browser used to access the web server on VM1 actually requested the default index.html file or when the browser provided it from its own cache. Likewise, precise details of each DNS query sent to the DNS server on VM3, and its corresponding responses, were retrieved. These results evidence that all the network traffic files generated respected the integrity and completeness of the data acquired.

For the previous, reasons H_3 was accepted, which means tcpdump is able to capture in real-time the network traffic of a VPS, respecting the integrity and completeness of the data acquired.

4.4 Summary

This chapter presented the results registered after completing the testing procedures described for each of the hypotheses stated. The chapter also interpreted the results to determine whether or not each hypothesis was accepted.

CHAPTER 5. DISCUSSION

The purpose of this study was to answer the following research question: is it possible to acquire forensically-sound digital evidence from a VPS hosted in a cloud provider's virtualization node that uses KVM as a hypervisor?

To address this question, three different hypotheses were stated:

H₁: images of the hard drives of a VPS can be created in an efficient manner, respecting the integrity and completeness of the data acquired.

H₂: images of the RAM content of a VPS can be created in an efficient manner, respecting the integrity and completeness of the data acquired.

H₃: the network traffic of a VPS can be captured in real-time, respecting the integrity and completeness of the data acquired.

In order to test the different hypotheses and to address the research question, a research environment was created to simulate a VPS provider's infrastructure. This environment consisted of a KVM virtualization node, which hosted four VMs acting as four independent VPSs.

The performance of two utilities to collect digital data from the VPSs was evaluated. The first one was virsh, a VMI out-of-VM and out-of-the-box utility, which was used to take snapshots of the virtual hard drives and to create RAM images of the VPSs. The second utility was tcpdump, which was used to capture in real-time the network traffic of the VPSs. In other words, the utility virsh was used to test H₁ and H₂ while tcpdump was used to test H₃.

The main reason for focusing on these utilities was they could positively impact on the admissibility of the evidence in a court of law because virsh works at the hypervisor level and tcpdump at the virtualization node OS level. This means the results of these utilities are more reliable than other utilities that work at the VM level (Dykstra & Sherman, 2012). Furthermore, they are available in the majority of the virtualization nodes based on GNU/Linux and their installation and execution do not require significant modifications to the production environment.

Once the research environment was prepared, a BASH script was developed to address H_1 by testing if the utility `virsh` was able to take external snapshots of the hard drives of the VPSs, from where bit-stream images could be created. After each image was successfully created, the snapshot was merged back into the original virtual hard drive. As it was mentioned in chapter four, the different tests executed showed efficiency, integrity, and completeness were achieved for each of the 56 different images generated. Therefore, H_1 was accepted.

Five different participants cooperated to address H_2 and H_3 by following four different scripts to interact with the four VPSs. During each interaction all the network traffic was captured with the utility `tcpdump`, and at the end of the interaction an image of the RAM content of the VPS was created with the utility `virsh`.

The examination of RAM images was performed using the Volatility framework and the results suggested integrity and completeness were achieved for each of the 40 RAM images generated. This means it was possible to recover from the RAM images specific actions performed by each participant. Efficiency was also achieved by every RAM imaging process executed. Therefore, H_2 was accepted.

Likewise, the examination of the network traffic captured was performed using Wireshark and the results suggested integrity and completeness were achieved for each of the 40 network traffic files generated. This entails it was possible to recover from the network traffic files specific actions performed by the participants. Therefore, H_3 was accepted.

The preceding results suggested that it is possible to acquire forensically-sound evidence from a VPS hosted in a cloud provider's virtualization node that uses KVM as a hypervisor because H_1 , H_2 , and H_3 were accepted. This means the utilities `virsh` and `tcpdump` are capable of collecting digital data from four VPSs in an efficient manner, respecting the integrity and completeness of the data acquired.

5.1 Procedure to acquire digital evidence from a VPS hosted in KVM

As it was mentioned, the results presented suggested it is possible to acquire forensically-sound digital evidence from a VPS hosted in a virtualization node that uses KVM as a hypervisor. These findings could be used to define a procedure to guide forensic practitioners in acquiring evidence in this environment. For instance, if a VPS named VM1 is suspected of being involved in a crime, the following processes could be executed in the KVM virtualization node to collect digital data from it:

- Create an image of each virtual hard drive assigned to VM1:

1. Take an external snapshot of each virtual hard drive:

```
# virsh snapshot-create-as VM1 --disk-only --atomic --name
snapshot1 --diskspec hda,snapshot=external,
file=/mnt/kvm/VM1-snapshot1.qcow2
```

When an external snapshot is created, the original virtual hard drive is set as read-only and a new overlay file is created to record the writing operations. This step is completed without suspending the VPS. The parameter `--diskspec` defines three options: the name of the virtual hard drive of the VPS to set as read-only (`hda`), the type of snapshot (`external`), and the new overlay file to be created (`/mnt/kvm/VM1-snapshot1.qcow2`). The parameter `--disk-only` specifies not to include the memory content in the snapshot. The parameter `--atomic` assures the snapshot either succeeds or fails with no changes on the original device. It is recommended to use `--atomic` every time an external snapshot is created, specially when it is created from a live system (Chirammal et al., 2016). Finally, the parameter `--name` specifies a customized name for the snapshot generated.

2. Create a bit-stream image from the original virtual hard drive, which is set as read-only after the external snapshot is created:

```
# dd if=/dev/lvm-group/VM1 of=/mnt/forensic-disk/VM1.dd
bs=512 conv=noerror,sync,fdatasync
```


3. Verify the bit-stream image is an exact and authentic copy of the original virtual hard drive by calculating MD5 and SHA1 hash values:

```
# md5sum /dev/lvm-group/VM1
# md5sum /mnt/forensic-disk/VM1.dd
# sha1sum /dev/lvm-group/VM1
# sha1sum /mnt/forensic-disk/VM1.dd
```

4. Merge the overlay file into the original virtual hard drive:

```
# virsh blockcommit VM1 hda --active --pivot --verbose
```

The parameter `hda` specifies the name of the virtual hard drive, the flag `--active` initiates the merging process of the overlay file into the virtual hard drive, and `--pivot` makes the hard drive active again once the merging is completed. At this point all the read/write operations are completed on the hard drive again and the overlay file is not used anymore. The flag `--verbose` displays detailed information about the process on the screen.

5. Remove the snapshot metadata and the overlay file:

```
# virsh snapshot-delete VM1 snapshot1 --metadata
# rm /mnt/kvm/VM1-snapshot1.qcow2
```

- Create an image of the RAM assigned to VM1:

1. Create a RAM image:

```
# virsh dump VM1 /mnt/forensic-disk/VM1.memdump --memory-only
```

The flag `--memory-only` specifies to collect only the RAM content and CPU common register value of the VPS. This command automatically suspends the VPS before creating the RAM image and resumes it after the image is completed. In case the VPS should not be suspended during the investigation, a RAM image can still be created if the flag `--live` is added to the command.

2. Verify if the RAM image generated can be examined using a memory forensic utility such as the Volatility Framework, specially if the image was generated with the flag --live.

- Capture in real-time the network traffic:

1. Start the network capturing process:

```
# tcpdump -nn -s0 --interface=vnet0 -w
/mnt/forensic-disk/VM1-vnet0.pcap
```

The parameter --interface specifies to capture only the traffic that traverses the listed network interfaces (vnet0). The parameter -nn declares not to resolve IP addresses (or ports) to host names (or services names). This option is relevant because the resolution process executed by default generates an important delay that can be omitted. The parameter -s indicates the snapshot length, which is the amount of bytes from each packet to be captured. The value 0 means no limit and it could be used to avoid truncating packets that have a longer size to the value specified. However, depending on the investigation and the legal constraints, this value could be reduced. For example, if the network is based on the Ethernet standard as data link layer, a value of 1,514 bytes should be enough because this is the maximum size of an Ethernet packet (Davidoff & Ham, 2012). Finally, the parameter -w declares the output file where the network traffic captured is stored.

2. Once the network capturing process is stopped, verify if the file generated can be examined using a network analysis utility such as Wireshark.

The previous procedure considers the utilities virsh and tcpdump are already installed on the virtualization node. The meaning of the arguments included in the example are described next:

- *VM1*: name of the VPS suspected of being involved in a crime. The names of all the VPSs hosted in a KVM virtualization node can be listed by executing the command: `virsh list --all`.

- *snapshot1*: customized name of the snapshot generated for VM1. The list of all the snapshots generated for this VPS can be displayed by executing the command: `virsh snapshot-list VM1`.
- *hda*: name of the virtual hard drive assigned to VM1. The name of all the virtual hard drives assigned to this VPS can be reported by executing the command: `virsh domblklist VM1`.
- *vnet0*: name of the TAP network interface assigned to VM1. The list of all the TAP interfaces assigned to this VPS can be displayed by executing the command: `virsh domiflist VM1`.
- */dev/lvm-group/VM1*: the device of the virtual hard drive assigned to VM1. The devices of all the virtual hard drives assigned to this VPS can be reported by executing the command: `virsh domblklist VM1`.
- */mnt/kvm/VM1-snapshot1.qcow2*: the name of the overlay file created when the external snapshot was taken, using the utility `virsh`.
- */mnt/forensic-disk/VM1.dd*: destination file of the bit-stream copy generated from the external snapshot, using the command `dd`.
- */mnt/forensic-disk/VM1.memdump*: destination file of the RAM image generated from VM1, using the utility `virsh`.
- */mnt/forensic-disk/VM1-vnet0.pcap*: destination file of the network traffic captured from VM1, using the utility `tcpdump`.

5.2 Significance

The findings of this study are important for the digital forensics field for several reasons. Firstly, Amazon Web Services (AWS), the largest IaaS cloud provider, recently announced a shift from Xen to KVM for future EC2 VPSs (TheRegister, 2017). Digital Ocean, the third largest cloud provider, uses KVM as a hypervisor (Chirammal et al.,

2016). These facts imply the usage of KVM will continue to grow in the near future and become the most preferred hypervisor by cloud providers. Furthermore, this research focused on different versions of Windows and GNU/Linux as OS for the VPSs. These two platforms represent the vast majority of the OS in the VPSs population and the utilities `virsh` and `tcpdump` were capable of successfully acquiring forensically-sound digital evidence from them. Even though `tcpdump` has been studied and used for a long time in the digital forensics field to capture network traffic on physical network interfaces, this study took a different approach and focused on using this utility to capture traffic on specific virtual network interfaces assigned to a VPS.

Secondly, the procedure defined in the section 5.1 could guide forensic practitioners and cloud providers to acquire forensically-sound digital evidence from a VPS (or any VM) involved in a crime and hosted in a virtualization node that uses KVM as a hypervisor. This procedure could be used as a base to develop a more detailed guideline and help forensic practitioners in acquiring evidence in this environment. In addition, the same methodology presented in this study could be used to analyze other utilities or hypervisors to verify if the results could be extended to include them. This research on cloud computing and virtual environments could contribute to enlarge the body of knowledge of digital forensics in this particular area and to reduce the criminal cases that involve the usage of VPSs.

Thirdly, forensic software companies could employ the findings of this research to develop remote agents to be executed in virtualization nodes that use KVM as a hypervisor in order to collect digital data from VPSs. If these agents were developed as open source projects, the details of their functioning would be known, which could potentially increase the willingness of the cloud providers to cooperate with the investigation and allow the execution of these agents inside their infrastructure. The agents could collect data locally or transfer it remotely using encryption mechanisms. The data collected could be imported into the existent forensic softwares suites to provide a unified examination interface to investigators. If similar research is conducted on other hypervisors, the remote agents could be improved to incorporate the new findings and collect digital data from VPSs hosted in these hypervisors.

Lastly, once the practical implications of the acquisition phase in distinct hypervisors are extensively analyzed, the global legislation on this area could be updated in order to deal with this problem also from a legal standpoint.

5.3 Limitations

This research was limited by time. The results and conclusions of this study were valid for a virtualization node based on a GNU/Linux Ubuntu Server 16.04.4 system (kernel version: 4.4.0-116.140). Ubuntu Server 18.04 (current LTS version) was released after this research began and for that reason the virtualization node was not based on it. Other operating systems or Linux kernel versions were not analyzed due to time constraints.

The KVM module used to convert the Linux kernel into a hypervisor was provided by the linux-image-kernel package (version 4.4.0-116.140) and the KVM user-space tools were provided by the qemu-kvm package (version 2.5.0). The libvirt (service and client) version used throughout the study was 1.3.1-1. Other versions of any of these packages were not analyzed in this study due to time constraints.

The utility virsh (version 1.3.1-1) was used to test H_1 and H_2 , while tcpdump (version 4.9.2-0) was used to test H_3 . No other utilities were employed to test the hypotheses also due to time constraints.

5.4 Recommendations for Future Studies

Future work on cloud computing and virtualization from a digital forensic perspective may consider extending this research to include the delimitations mentioned in chapter one. For instance, the testing procedures in this study were completed under low load conditions on the virtualization node. This means the four VMs and the virtualization node did not execute heavy processes during the creation of virtual hard drive and RAM content images and during the network traffic capturing processes. It would be interesting to analyze how the efficiency of these procedures is affected under heavy load conditions

on the virtualization node. If efficiency is notably affected, the live migration support provided by KVM could be studied to determine if a VM can be migrated to a special virtualization node (with more resources and low load) in order to complete the acquisition process in that node. This approach may be also beneficial from a privacy point of view, since the virtualization node could host only the VM under examination.

Even though this study was focused on VPSs, the methodology and techniques presented could be employed to acquire digital data from any VM hosted in a virtualization node that uses KVM as a hypervisor. This methodology could also be used to define best practice guidelines to help digital investigators to perform acquisitions on this environment. Future work could take into account other OS to install on the virtualization node, other utilities instead of `virsh` and `tcpdump`, or different versions of KVM (kernel module or user-space tools) or `libvirt` (service or client) to verify if the results could be extended to include them.

The methodology presented could also be used to create a virtual environment with KVM in order to analyze the behavior of a piece of malware (or other type software) after it is executed in a VM. The usage of `virsh` to create RAM content images and `tcpdump` to capture all the network traffic in real-time could be valuable resources for digital investigators to inspect the behavior of a particular piece of software.

Finally, it is worth mentioning briefly two points about the procedure employed to test the creation of virtual hard drive images. First, a block size of 512 bytes was defined every time the command `dd` was executed in this study. The objective of defining this value was to match the physical sector size of both hard drives, which was also 512 bytes. It would be valuable to analyze the impact defining a longer sector size could have on efficiency when the command `dd` is executed. Second, the BASH script developed to generate automatically bit-stream images from the virtual hard drives created two independent images: the first one was stored into a local file in a partition formatted with `ext4`, and the second one was written directly to physical sectors of a different partition. This study only included the time results observed for the second image created because the first image always reported transfer rates higher than the baseline. This difference was generated because the filesystem (`ext4`) noticed when a segment of consecutive sectors

was full of zeros and it used this knowledge to omit writing these sectors. This feature provided by the filesystem increased notably the performance of the imaging process. Even though this finding could be interesting and beneficial in other circumstances, it was not a precise value to measure efficiency for the purpose of this research and it was not taken into account.

5.5 Summary

This chapter drew conclusions from the results reported in chapter four in order to address the research question. It also presented a procedure to acquire forensically-sound digital evidence from a VPS hosted in a KVM virtualization node. Finally, the chapter analyzed the significance of the findings for the digital forensics field and stated limitations and recommendations for future related studies.

REFERENCES

- BankInfoSecurity. (2017). *Crime as a service: A top cyber threat for 2017*. Retrieved from <https://www.bankinfosecurity.com/>.
- Barrett, D., & Kipper, G. (2010). *Virtualization and forensics: A digital forensic investigators guide to virtual environments*. Syngress.
- Beebe, N. (2009). The good, the bad and the unaddressed. In S. Shenoit & G. Peterson (Eds.), *Advances in digital forensics V* (p. 17-36). Springer Science & Business Media.
- Bem, D., & Huebner, E. (2007). Computer forensic analysis in a virtual environment. *International journal of digital evidence*, 6(2), 1-13.
- Birk, D. (2011, January). Technical challenges of forensic investigations in cloud computing environments. In *Workshop on cryptography and security in clouds*. Zurich, Switzerland.
- Birk, D., & Wegener, C. (2011, May). Technical issues of forensic investigations in cloud computing environments. In *Proceedings of the 6th international workshop on systematic approaches to digital forensic engineering (SADFE)* (p. 1-10). Oakland, CA: IEEE.
- Chiramal, H. D., Mukhedkar, P., & Vettathu, A. (2016). *Mastering KVM virtualization*. Packt Publishing.
- Davidoff, S., & Ham, J. (2012). *Network forensics: tracking hackers through cyberspace*. Upper Saddle River, NJ: Prentice Hall.
- Dolan-Gavitt, B., Payne, B., & Lee, W. (2011). *Leveraging forensic tools for virtual machine introspection*. Georgia Institute of Technology.

- 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, 90-98.
- Forbes. (2016). *Roundup of cloud computing forecasts and market estimates*. Retrieved from <https://www.forbes.com/>.
- Garcia, M. A. (2014). Cloud computing forensics. In S. Srinivasan (Ed.), *Security, trust, and regulatory aspects of cloud computing in business environments* (p. 170-178). IGI Global.
- Hebbal, Y., Laniece, S., & Menaud, J. M. (2015, August). Virtual machine introspection: Techniques and applications. In *Proceedings of the 10th international conference on availability, reliability and security (ARES)* (p. 676-685). IEEE.
- Holt, T. J., Bossler, A. M., & Seigfried-Spellar, K. C. (2015). *Cybercrime and digital forensics: An introduction*. New York, NY: Routledge.
- IBM. (2018). *Best practices for vm storage devices*. Retrieved from <https://www.ibm.com/support/>.
- ITProPortal. (2011). *Hackers used Amazon's EC2 cloud service to launch attack on Playstation network*. Retrieved from <https://www.itproportal.com/>.
- Kolhe, M., & Ahirao, P. (2017). Live vs dead computer forensic image acquisition. *International journal of computer science and information technologies (IJCSIT)*, 8(3), 455-457.
- Lessing, M., & von Solms, B. (2008). *Live forensic acquisition as alternative to traditional forensic processes*. Retrieved from <https://www.researchgate.net/>.
- Libvirt. (2018). *Storage management*. Retrieved from <https://libvirt.org/>.

- Ligh, M. H., Case, A., Levy, J., & Walters, A. (2014). *The art of memory forensics: Detecting malware and threats in Windows, Linux, and Mac memory*. Indianapolis, IN: John Wiley & Sons.
- McKemmish, R. (2008). When is digital evidence forensically sound? In I. Ray & S. Sheno (Eds.), *Advances in digital forensics IV* (p. 3-15). Springer Science & Business Media.
- Mell, P., & Grance, T. (2011). *The NIST definition of cloud computing*.
- Morioka, E., & Sharbaf, M. S. (2016, May). Digital forensics research on cloud computing: An investigation of cloud forensics solutions. In *Symposium on technologies for homeland security (HST)* (p. 1-6). IEEE.
- Owen, P., & Thomas, P. (2011). An analysis of digital forensic examinations: Mobile devices versus hard disk drives utilising ACPO & NIST guidelines. *Digital investigation*, 8(2), 135-140.
- Patrascu, A., & Patriciu, V. V. (2014). Logging system for cloud computing forensic environments. *Journal of control engineering and applied informatics*, 16(1), 80-88.
- Poisel, R., Malzer, E., & Tjoa, S. (2013). Evidence and cloud computing: The virtual machine introspection approach. *Journal of wireless mobile networks, ubiquitous computing, and dependable applications (JoWUA)*, 4(1), 135-152.
- Pollitt, M., Nance, K., Hay, B., Dodge, R. C., Craiger, P., Burke, P., ... Brubaker, B. (2008). Virtualization and digital forensics: A research and education agenda. *Journal of digital forensic practice*, 2(2), 62-73.
- RedHat. (2015). *Red hat enterprise virtualization 3.5 technical guide*. Retrieved from <https://access.redhat.com/documentation/>.

RedHat. (2017). *Red hat enterprise linux 7: Virtualization getting started guide*.

Retrieved from <https://access.redhat.com/documentation/>.

RedHat. (2018). *Red hat virtualization 4.1: Technical reference*. Retrieved from

<https://access.redhat.com/documentation/>.

Reilly, D., Wren, C., & Berry, T. (2010, November). Cloud computing: Forensic challenges for law enforcement. In *International conference for internet technology and secured transactions ICITST* (p. 1-7). IEEE.

RightScale. (2016). *Cloud computing trends: 2016 state of the cloud survey*. Retrieved from <https://www.rightscale.com/>.

Rogers, M. K. (2017). Technology and digital forensics. In M. R. McGuire & T. J. Holt (Eds.), *The routledge handbook of technology, crime and justice* (p. 406-416). New York, NY: Routledge.

Rogers, M. K., Goldman, J., Mislán, R., Wedge, T., & Debrota, S. (2006). Computer forensics field triage process model. *Journal of Digital Forensics, Security and Law*, 1(2), 19-38.

Sang, T. (2013, January). A log based approach to make digital forensics easier on cloud computing. In *Third international conference on intelligent system design and engineering applications (ISDEA)* (p. 91-94). IEEE.

Suneja, S., Isci, C., & de Lara, E. (2015, March). Exploring VM introspection: Techniques and trade-offs. *ACM Sigplan Notices*, 50(7), 133-146.

SynergyResearchGroup. (2016). *2015 review shows \$110 billion cloud market growing at 28% annually*. Retrieved from <https://www.srgresearch.com/>.

Technavio. (2015). *Virtualization is prompting impressive growth in cloud-enabling technologies*. Retrieved from <https://www.technavio.com/>.

TheRegister. (2017). *AWS adopts home-brewed kvm as new hypervisor*. Retrieved from <https://www.theregister.co.uk/>.

Volatility. (2018). *About the Volatility foundation*. Retrieved from <http://www.volatilityfoundation.org/>.

WireShark. (2018). *About Wireshark*. Retrieved from <https://www.wireshark.org/>.

Zahedi, S. (2014). *Virtualization security threat forensic and environment safeguarding*. Retrieved from <http://www.diva-portal.org/>.

Zawoad, S., & Hasan, R. (2013, February). Cloud forensics: A meta-study of challenges, approaches, and open problems. Cornell University Library, arXiv:1302.6312.

ZDNet. (2012). *Amazon EC2 cloud is made up of almost half-a-million linux servers*. Retrieved from <http://www.zdnet.com/>.

APPENDIX A. PACKAGES INSTALLED ON THE VIRTUALIZATION NODE

Packages installed on the virtualization node and their respective version.

Package name	Version
accountsservice	0.6.40-2ubuntu11.3
acl	2.2.52-3
acpid	1:2.0.26-1ubuntu2
adduser	3.113+nmu3ubuntu4
apparmor	2.10.95-0ubuntu2.8
apport	2.20.1-0ubuntu2.15
apport-symptoms	0.20
apt	1.2.25
apt-transport-https	1.2.25
apt-utils	1.2.25
at	3.1.18-2ubuntu1
augeas-lenses	1.4.0-0ubuntu1.1
base-files	9.4ubuntu4.6
base-passwd	3.5.39
bash	4.3-14ubuntu1.2
bash-completion	1:2.1-4.2ubuntu1.1
bcache-tools	1.0.8-2
bind9-host	1:9.10.3.dfsg.P4-8ubuntu1.10
binutils	2.26.1-1ubuntu1~16.04.6
bridge-utils	1.5-9ubuntu1
bsdmainutils	9.0.6ubuntu3
bsdutils	1:2.27.1-6ubuntu3.4
btrfs-tools	4.4-1ubuntu1
busybox-initramfs	1:1.22.0-15ubuntu1
busybox-static	1:1.22.0-15ubuntu1
byobu	5.106-0ubuntu1
bzip2	1.0.6-8
ca-certificates	20170717~16.04.1
cgmanager	0.39-2ubuntu5
cloud-guest-utils	0.27-0ubuntu25
cloud-initramfs-copymods	0.27ubuntu1.5
cloud-initramfs-dyn-netconf	0.27ubuntu1.5
command-not-found	0.3ubuntu16.04.2
command-not-found-data	0.3ubuntu16.04.2
console-setup	1.108ubuntu15.3
console-setup-linux	1.108ubuntu15.3

coreutils	8.25-2ubuntu3~16.04
cpio	2.11+dfsg-5ubuntu1
cpu-checker	0.7-0ubuntu7
crda	3.13-1
cron	3.0pl1-128ubuntu2
cryptsetup	2:1.6.6-5ubuntu2.1
cryptsetup-bin	2:1.6.6-5ubuntu2.1
curl	7.47.0-1ubuntu2.6
dash	0.5.8-2.1ubuntu2
dbus	1.10.6-1ubuntu3.3
debconf	1.5.58ubuntu1
debconf-i18n	1.5.58ubuntu1
debianutils	4.7
dh-python	2.20151103ubuntu1.1
diffutils	1:3.3-3
distro-info-data	0.28ubuntu0.7
dmeventd	2:1.02.110-1ubuntu10
dmidecode	3.0-2ubuntu0.1
dmsetup	2:1.02.110-1ubuntu10
dns-root-data	2015052300+h+1
dnsmasq-base	2.75-1ubuntu0.16.04.4
dnsutils	1:9.10.3.dfsg.P4-8ubuntu1.10
dosfstools	3.0.28-2ubuntu0.1
dpkg	1.18.4ubuntu1.3
e2fslibs:amd64	1.42.13-1ubuntu1
e2fsprogs	1.42.13-1ubuntu1
ebtables	2.0.10.4-3.4ubuntu2
ed	1.10-2
efibootmgr	0.12-4
eject	2.1.5+deb1+cvs20081104-13.1ubuntu0.16.04.1
ethtool	1:4.5-1
exfat-fuse	1.2.3-1
exfat-utils	1.2.3-1
file	1:5.25-2ubuntu1
findutils	4.6.0+git+20160126-2
fonts-ubuntu-font-family-console	1:0.83-0ubuntu2
friendly-recovery	0.2.31ubuntu1
ftp	0.17-33
fuse	2.9.4-1ubuntu3.1
gawk	1:4.1.3+dfsg-0.1
gcc-5-base:amd64	5.4.0-6ubuntu1~16.04.9
gcc-6-base:amd64	6.0.1-0ubuntu1
gdisk	1.0.1-1build1
geoip-database	20160408-1
gettext-base	0.19.7-2ubuntu3

gir1.2-glib-2.0:amd64	1.46.0-3ubuntu1
git	1:2.7.4-0ubuntu1.3
git-man	1:2.7.4-0ubuntu1.3
gnupg	1.4.20-1ubuntu3.1
gpgv	1.4.20-1ubuntu3.1
grep	2.25-1~16.04.1
groff-base	1.22.3-7
grub-common	2.02~beta2-36ubuntu3.17
grub-efi-amd64	2.02~beta2-36ubuntu3.17
grub-efi-amd64-bin	2.02~beta2-36ubuntu3.17
grub-efi-amd64-signed	1.66.17+2.02~beta2-36ubuntu3.17
grub-legacy-ec2	17.2-35-gf576b2a2-0ubuntu1~16.04.2
grub2-common	2.02~beta2-36ubuntu3.17
gzip	1.6-4ubuntu1
hdparm	9.48+ds-1
hostname	3.16ubuntu2
ifenslave	2.7ubuntu1
ifupdown	0.8.10ubuntu1.2
info	6.1.0.dfsg.1-5
init	1.29ubuntu4
init-system-helpers	1.29ubuntu4
initramfs-tools	0.122ubuntu8.10
initramfs-tools-bin	0.122ubuntu8.10
initramfs-tools-core	0.122ubuntu8.10
initscripts	2.88dsf-59.3ubuntu2
insserv	1.14.0-5ubuntu3
install-info	6.1.0.dfsg.1-5
installation-report	2.60ubuntu1
iproute2	4.3.0-1ubuntu3.16.04.3
iptables	1.6.0-2ubuntu3
iputils-ping	3:20121221-5ubuntu2
iputils-tracepath	3:20121221-5ubuntu2
ipxe-qemu	1.0.0+git-20150424.a25a16d-1ubuntu1.2
irqbalance	1.1.0-2ubuntu1
isc-dhcp-client	4.3.3-5ubuntu12.9
isc-dhcp-common	4.3.3-5ubuntu12.9
iso-codes	3.65-1
iw	3.17-1
kbd	1.15.5-1ubuntu5
keyboard-configuration	1.108ubuntu15.3
klibc-utils	2.0.4-8ubuntu1.16.04.4
kmod	22-1ubuntu5
kpartx	0.5.0+git1.656f8865-5ubuntu2.5
krb5-locales	1.13.2+dfsg-5ubuntu2
language-selector-common	0.165.4

laptop-detect	0.13.7ubuntu2
less	481-2.1ubuntu0.2
libaccountsservice0:amd64	0.6.40-2ubuntu11.3
libacl1:amd64	2.2.52-3
libaio1:amd64	0.3.110-2
libapparmor-perl	2.10.95-0ubuntu2.8
libapparmor1:amd64	2.10.95-0ubuntu2.8
libapt-inst2.0:amd64	1.2.25
libapt-pkg5.0:amd64	1.2.25
libasn1-8-heimdal:amd64	1.7~git20150920+dfsg-4ubuntu1.16.04.1
libasound2:amd64	1.1.0-0ubuntu1
libasound2-data	1.1.0-0ubuntu1
libasprintf0v5:amd64	0.19.7-2ubuntu3
libasyncns0:amd64	0.8-5build1
libatm1:amd64	1:2.5.1-1.5
libattr1:amd64	1:2.4.47-2
libaudit-common	1:2.4.5-1ubuntu2.1
libaudit1:amd64	1:2.4.5-1ubuntu2.1
libaugeas0	1.4.0-0ubuntu1.1
libavahi-client3:amd64	0.6.32~rc+dfsg-1ubuntu2
libavahi-common-data:amd64	0.6.32~rc+dfsg-1ubuntu2
libavahi-common3:amd64	0.6.32~rc+dfsg-1ubuntu2
libbind9-140:amd64	1:9.10.3.dfsg.P4-8ubuntu1.10
libblkid1:amd64	2.27.1-6ubuntu3.4
libbluetooth3:amd64	5.37-0ubuntu5.1
libboost-iostreams1.58.0:amd64	1.58.0+dfsg-5ubuntu3.1
libboost-random1.58.0:amd64	1.58.0+dfsg-5ubuntu3.1
libboost-system1.58.0:amd64	1.58.0+dfsg-5ubuntu3.1
libboost-thread1.58.0:amd64	1.58.0+dfsg-5ubuntu3.1
libbrlapi0.6:amd64	5.3.1-2ubuntu2.1
libbsd0:amd64	0.8.2-1
libbz2-1.0:amd64	1.0.6-8
libc-bin	2.23-0ubuntu10
libc6:amd64	2.23-0ubuntu10
libcaca0:amd64	0.99.beta19-2build2~gcc5.2
libcacard0:amd64	1:2.5.0-2
libcap-ng0:amd64	0.7.7-1
libcap2:amd64	1:2.24-12
libcap2-bin	1:2.24-12
libcgmanager0:amd64	0.39-2ubuntu5
libcomerr2:amd64	1.42.13-1ubuntu1
libcryptsetup4:amd64	2:1.6.6-5ubuntu2.1
libcurl3-gnutls:amd64	7.47.0-1ubuntu2.6
libdb5.3:amd64	5.3.28-11ubuntu0.1
libdbus-1-3:amd64	1.10.6-1ubuntu3.3

libdbus-glib-1-2:amd64	0.106-1
libdebconfclient0:amd64	0.198ubuntu1
libdevmapper-event1.02.1:amd64	2:1.02.110-1ubuntu10
libdevmapper1.02.1:amd64	2:1.02.110-1ubuntu10
libdistorm3-3	3.3.0-3
libdns-export162	1:9.10.3.dfsg.P4-8ubuntu1.10
libdns162:amd64	1:9.10.3.dfsg.P4-8ubuntu1.10
libdrm-common	2.4.83-1~16.04.1
libdrm2:amd64	2.4.83-1~16.04.1
libdumbnet1:amd64	1.12-7
libedit2:amd64	3.1-20150325-1ubuntu2
libefivar0:amd64	0.23-2
libelf1:amd64	0.165-3ubuntu1
liberror-perl	0.17-1.2
libestr0	0.1.10-1
libevent-2.0-5:amd64	2.0.21-stable-2ubuntu0.16.04.1
libexpat1:amd64	2.1.0-7ubuntu0.16.04.3
libfdisk1:amd64	2.27.1-6ubuntu3.4
libfdt1:amd64	1.4.0+dfsg-2
libffi6:amd64	3.2.1-4
libflac8:amd64	1.3.1-4
libfreetype6:amd64	2.6.1-0.1ubuntu2.3
libfribidi0:amd64	0.19.7-1
libfuse2:amd64	2.9.4-1ubuntu3.1
libgcc1:amd64	1:6.0.1-0ubuntu1
libgcrypt20:amd64	1.6.5-2ubuntu0.3
libgdbm3:amd64	1.8.3-13.1
libgeoip1:amd64	1.6.9-1
libgirepository-1.0-1:amd64	1.46.0-3ubuntu1
libglib2.0-0:amd64	2.48.2-0ubuntu1
libglib2.0-data	2.48.2-0ubuntu1
libgmp10:amd64	2:6.1.0+dfsg-2
libgnutls-openssl27:amd64	3.4.10-4ubuntu1.4
libgnutls30:amd64	3.4.10-4ubuntu1.4
libgpg-error0:amd64	1.21-2ubuntu1
libgpm2:amd64	1.20.4-6.1
libgssapi-krb5-2:amd64	1.13.2+dfsg-5ubuntu2
libgssapi3-heimdal:amd64	1.7~git20150920+dfsg-4ubuntu1.16.04.1
libhcrypto4-heimdal:amd64	1.7~git20150920+dfsg-4ubuntu1.16.04.1
libheimbase1-heimdal:amd64	1.7~git20150920+dfsg-4ubuntu1.16.04.1
libheimntlm0-heimdal:amd64	1.7~git20150920+dfsg-4ubuntu1.16.04.1
libhogweed4:amd64	3.2-1ubuntu0.16.04.1
libhx509-5-heimdal:amd64	1.7~git20150920+dfsg-4ubuntu1.16.04.1
libicu55:amd64	55.1-7ubuntu0.3
libidn11:amd64	1.32-3ubuntu1.2

libisc-export160	1:9.10.3.dfsg.P4-8ubuntu1.10
libisc160:amd64	1:9.10.3.dfsg.P4-8ubuntu1.10
libisccc140:amd64	1:9.10.3.dfsg.P4-8ubuntu1.10
libiscconf140:amd64	1:9.10.3.dfsg.P4-8ubuntu1.10
libiscsi2:amd64	1.12.0-2
libjpeg-turbo8:amd64	1.4.2-0ubuntu3
libjpeg8:amd64	8c-2ubuntu8
libjson-c2:amd64	0.11-4ubuntu2
libk5crypto3:amd64	1.13.2+dfsg-5ubuntu2
libkeyutils1:amd64	1.5.9-8ubuntu1
libklibc	2.0.4-8ubuntu1.16.04.4
libkmod2:amd64	22-1ubuntu5
libkrb5-26-heimdal:amd64	1.7~git20150920+dfsg-4ubuntu1.16.04.1
libkrb5-3:amd64	1.13.2+dfsg-5ubuntu2
libkrb5support0:amd64	1.13.2+dfsg-5ubuntu2
libldap-2.4-2:amd64	2.4.42+dfsg-2ubuntu3.2
liblocale-gettext-perl	1.07-1build1
liblvm2app2.2:amd64	2.02.133-1ubuntu10
liblvm2cmd2.02:amd64	2.02.133-1ubuntu10
liblwres141:amd64	1:9.10.3.dfsg.P4-8ubuntu1.10
liblxc1	2.0.8-0ubuntu1~16.04.2
liblz4-1:amd64	0.0~r131-2ubuntu2
liblzma5:amd64	5.1.1alpha+20120614-2ubuntu2
liblzo2-2:amd64	2.08-1.2
libmagic1:amd64	1:5.25-2ubuntu1
libmnl0:amd64	1.0.3-5
libmount1:amd64	2.27.1-6ubuntu3.4
libmpdec2:amd64	2.4.2-1
libmpfr4:amd64	3.1.4-1
libmspack0:amd64	0.5-1ubuntu0.16.04.1
libncurses5:amd64	6.0+20160213-1ubuntu1
libncursesw5:amd64	6.0+20160213-1ubuntu1
libnetcf1:amd64	1:0.2.8-1ubuntu1
libnetfilter-contrack3:amd64	1.0.5-1
libnettle6:amd64	3.2-1ubuntu0.16.04.1
libnewt0.52:amd64	0.52.18-1ubuntu2
libnfnetwork0:amd64	1.0.1-3
libnih-dbus1:amd64	1.0.3-4.3ubuntu1
libnih1:amd64	1.0.3-4.3ubuntu1
libnl-3-200:amd64	3.2.27-1ubuntu0.16.04.1
libnl-genl-3-200:amd64	3.2.27-1ubuntu0.16.04.1
libnl-route-3-200:amd64	3.2.27-1ubuntu0.16.04.1
libnspr4:amd64	2:4.13.1-0ubuntu0.16.04.1
libnss3:amd64	2:3.28.4-0ubuntu0.16.04.3
libnss3-nssdb	2:3.28.4-0ubuntu0.16.04.3

libnuma1:amd64	2.0.11-1ubuntu1.1
libogg0:amd64	1.3.2-1
libopts25:amd64	1:5.18.7-3
libopus0:amd64	1.1.2-1ubuntu1
libp11-kit0:amd64	0.23.2-5~ubuntu16.04.1
libpam-modules:amd64	1.1.8-3.2ubuntu2
libpam-modules-bin	1.1.8-3.2ubuntu2
libpam-runtime	1.1.8-3.2ubuntu2
libpam-systemd:amd64	229-4ubuntu21.1
libpam0g:amd64	1.1.8-3.2ubuntu2
libparted2:amd64	3.2-15ubuntu0.1
libpcap0.8:amd64	1.7.4-2
libpci3:amd64	1:3.3.1-1.1ubuntu1.1
libpciaccess0:amd64	0.13.4-1
libpcre3:amd64	2:8.38-3.1
libperl5.22:amd64	5.22.1-9ubuntu0.2
libpipeline1:amd64	1.4.1-2
libpixman-1-0:amd64	0.33.6-1
libplymouth4:amd64	0.9.2-3ubuntu13.2
libpng12-0:amd64	1.2.54-1ubuntu1
libpolkit-agent-1-0:amd64	0.105-14.1
libpolkit-backend-1-0:amd64	0.105-14.1
libpolkit-gobject-1-0:amd64	0.105-14.1
libpopt0:amd64	1.16-10
libprocps4:amd64	2:3.3.10-4ubuntu2.3
libpulse0:amd64	1:8.0-0ubuntu3.8
libpython-stdlib:amd64	2.7.12-1~16.04
libpython2.7-minimal:amd64	2.7.12-1ubuntu0~16.04.3
libpython2.7-stdlib:amd64	2.7.12-1ubuntu0~16.04.3
libpython3-stdlib:amd64	3.5.1-3
libpython3.5:amd64	3.5.2-2ubuntu0~16.04.4
libpython3.5-minimal:amd64	3.5.2-2ubuntu0~16.04.4
libpython3.5-stdlib:amd64	3.5.2-2ubuntu0~16.04.4
librados2	10.2.9-0ubuntu0.16.04.1
librbd1	10.2.9-0ubuntu0.16.04.1
libreadline5:amd64	5.2+dfsg-3build1
libreadline6:amd64	6.3-8ubuntu2
libroken18-heimdal:amd64	1.7~git20150920+dfsg-4ubuntu1.16.04.1
librtmp1:amd64	2.4+20151223.gitfa8646d-1ubuntu0.1
libsasl2-2:amd64	2.1.26.dfsg1-14build1
libsasl2-modules:amd64	2.1.26.dfsg1-14build1
libsasl2-modules-db:amd64	2.1.26.dfsg1-14build1
libsdl1.2debian:amd64	1.2.15+dfsg1-3
libseccomp2:amd64	2.3.1-2.1ubuntu2~16.04.1
libselinux1:amd64	2.4-3build2

libsemanage-common	2.3-1build3
libsemanage1:amd64	2.3-1build3
libsepol1:amd64	2.4-2
libsigsegv2:amd64	2.10-4
libslang2:amd64	2.3.0-2ubuntu1
libsmartcols1:amd64	2.27.1-6ubuntu3.4
libsndfile1:amd64	1.0.25-10ubuntu0.16.04.1
libspice-server1:amd64	0.12.6-4ubuntu0.3
libsqlite3-0:amd64	3.11.0-1ubuntu1
libss2:amd64	1.42.13-1ubuntu1
libssh2-1:amd64	1.5.0-2ubuntu0.1
libssl1.0.0:amd64	1.0.2g-1ubuntu4.10
libstdc++6:amd64	5.4.0-6ubuntu1~16.04.9
libsystemd0:amd64	229-4ubuntu21.1
libtasn1-6:amd64	4.7-3ubuntu0.16.04.3
libtext-charwidth-perl	0.04-7build5
libtext-iconv-perl	1.7-5build4
libtext-wrapi18n-perl	0.06-7.1
libtinfo5:amd64	6.0+20160213-1ubuntu1
libudev1:amd64	229-4ubuntu21.1
libusb-0.1-4:amd64	2:0.1.12-28
libusb-1.0-0:amd64	2:1.0.20-1
libusbredirparser1:amd64	0.7.1-1
libustr-1.0-1:amd64	1.0.4-5
libutempter0:amd64	1.1.6-3
libuuid1:amd64	2.27.1-6ubuntu3.4
libvirt-bin	1.3.1-1ubuntu10.19
libvirt0:amd64	1.3.1-1ubuntu10.19
libvorbis0a:amd64	1.3.5-3ubuntu0.1
libvorbisenc2:amd64	1.3.5-3ubuntu0.1
libwind0-heimdal:amd64	1.7~git20150920+dfsg-4ubuntu1.16.04.1
libwrap0:amd64	7.6.q-25
libx11-6:amd64	2:1.6.3-1ubuntu2
libx11-data	2:1.6.3-1ubuntu2
libx86-1:amd64	1.1+ds1-10
libxau6:amd64	1:1.0.8-1
libxcb1:amd64	1.11.1-1ubuntu1
libxdmcp6:amd64	1:1.1.2-1.1
libxen-4.6:amd64	4.6.5-0ubuntu1.4
libxenstore3.0:amd64	4.6.5-0ubuntu1.4
libxext6:amd64	2:1.3.3-1
libxml2:amd64	2.9.3+dfsg1-1ubuntu0.5
libxml2-utils	2.9.3+dfsg1-1ubuntu0.5
libxmuu1:amd64	2:1.1.2-2
libxslt1.1:amd64	1.1.28-2.1ubuntu0.1

libxtables11:amd64	1.6.0-2ubuntu3
libyajl2:amd64	2.1.0-2
linux-base	4.0ubuntu1
linux-firmware	1.157.17
linux-headers-4.4.0-112	4.4.0-112.135
linux-headers-4.4.0-112-generic	4.4.0-112.135
linux-headers-4.4.0-116	4.4.0-116.140
linux-headers-4.4.0-116-generic	4.4.0-116.140
linux-headers-4.4.0-62	4.4.0-62.83
linux-headers-4.4.0-62-generic	4.4.0-62.83
linux-headers-generic	4.4.0.116.122
linux-image-4.4.0-112-generic	4.4.0-112.135
linux-image-4.4.0-116-generic	4.4.0-116.140
linux-image-4.4.0-62-generic	4.4.0-62.83
linux-image-extra-4.4.0-112-generic	4.4.0-112.135
linux-image-extra-4.4.0-116-generic	4.4.0-116.140
linux-image-extra-4.4.0-62-generic	4.4.0-62.83
linux-signed-generic	4.4.0.116.122
linux-signed-image-4.4.0-112-generic	4.4.0-112.135
linux-signed-image-4.4.0-116-generic	4.4.0-116.140
linux-signed-image-4.4.0-62-generic	4.4.0-62.83
linux-signed-image-generic	4.4.0.116.122
locales	2.23-0ubuntu10
login	1:4.2-3.1ubuntu5.3
logrotate	3.8.7-2ubuntu2.16.04.2
lsb-base	9.20160110ubuntu0.2
lsb-release	9.20160110ubuntu0.2
lshw	02.17-1.1ubuntu3.4
lsof	4.89+dfsg-0.1
ltrace	0.7.3-5.1ubuntu4
lvm2	2.02.133-1ubuntu10
lxc-common	2.0.8-0ubuntu1~16.04.2
lxcfs	2.0.8-0ubuntu1~16.04.2
lxd	2.0.11-0ubuntu1~16.04.4
lxd-client	2.0.11-0ubuntu1~16.04.4
makedev	2.3.1-93ubuntu2~ubuntu16.04.1
man-db	2.7.5-1
manpages	4.04-2
mawk	1.3.3-17ubuntu2
mc	3:4.8.15-2
mc-data	3:4.8.15-2
mdadm	3.3-2ubuntu7.6
memtest86+	5.01-3ubuntu2
mime-support	3.59ubuntu1
mlocate	0.26-1ubuntu2

mokutil	0.3.0-0ubuntu3
mount	2.27.1-6ubuntu3.4
msr-tools	1.3-2
mtr-tiny	0.86-1ubuntu0.1
multiarch-support	2.23-0ubuntu10
nano	2.5.3-2ubuntu2
ncurses-base	6.0+20160213-1ubuntu1
ncurses-bin	6.0+20160213-1ubuntu1
ncurses-term	6.0+20160213-1ubuntu1
net-tools	1.60-26ubuntu1
netbase	5.3
netcat-openbsd	1.105-7ubuntu1
ntfs-3g	1:2015.3.14AR.1-1ubuntu0.1
ntp	1:4.2.8p4+dfsg-3ubuntu5.8
open-iscsi	2.0.873+git0.3b4b4500-14ubuntu3.4
open-vm-tools	2:10.0.7-3227872-5ubuntu1~16.04.2
openssh-client	1:7.2p2-4ubuntu2.4
openssh-server	1:7.2p2-4ubuntu2.4
openssh-sftp-server	1:7.2p2-4ubuntu2.4
openssl	1.0.2g-1ubuntu4.10
os-prober	1.70ubuntu3.3
overlayroot	0.27ubuntu1.5
parted	3.2-15ubuntu0.1
passwd	1:4.2-3.1ubuntu5.3
pastebinit	1.5-1
patch	2.7.5-1
pciutils	1:3.3.1-1.1ubuntu1.1
perl	5.22.1-9ubuntu0.2
perl-base	5.22.1-9ubuntu0.2
perl-modules-5.22	5.22.1-9ubuntu0.2
plymouth	0.9.2-3ubuntu13.2
plymouth-theme-ubuntu-text	0.9.2-3ubuntu13.2
pm-utils	1.4.1-16
policykit-1	0.105-14.1
popularity-contest	1.64ubuntu2
powermgmt-base	1.31+nmu1
procps	2:3.3.10-4ubuntu2.3
psmisc	22.21-2.1build1
python	2.7.12-1~16.04
python-apt-common	1.1.0~beta1ubuntu0.16.04.1
python-crypto	2.6.1-6ubuntu0.16.04.2
python-distorm3	3.3.0-3
python-minimal	2.7.12-1~16.04
python2.7	2.7.12-1ubuntu0~16.04.3
python2.7-minimal	2.7.12-1ubuntu0~16.04.3

python3	3.5.1-3
python3-apport	2.20.1-0ubuntu2.15
python3-apt	1.1.0~beta1ubuntu0.16.04.1
python3-chardet	2.3.0-2
python3-commandnotfound	0.3ubuntu16.04.2
python3-dbus	1.2.0-3
python3-debian	0.1.27ubuntu2
python3-distupgrade	1:16.04.24
python3-gdbm:amd64	3.5.1-1
python3-gi	3.20.0-0ubuntu1
python3-minimal	3.5.1-3
python3-newt	0.52.18-1ubuntu2
python3-pkg-resources	20.7.0-1
python3-problem-report	2.20.1-0ubuntu2.15
python3-pycurl	7.43.0-1ubuntu1
python3-requests	2.9.1-3
python3-six	1.10.0-3
python3-software-properties	0.96.20.7
python3-systemd	231-2build1
python3-update-manager	1:16.04.12
python3-urllib3	1.13.1-2ubuntu0.16.04.1
python3.5	3.5.2-2ubuntu0~16.04.4
python3.5-minimal	3.5.2-2ubuntu0~16.04.4
qemu-block-extra:amd64	1:2.5+dfsg-5ubuntu10.24
qemu-kvm	1:2.5+dfsg-5ubuntu10.24
qemu-system-common	1:2.5+dfsg-5ubuntu10.24
qemu-system-x86	1:2.5+dfsg-5ubuntu10.24
qemu-utils	1:2.5+dfsg-5ubuntu10.24
readline-common	6.3-8ubuntu2
rename	0.20-4
resolvconf	1.78ubuntu6
rsync	3.1.1-3ubuntu1.2
rsyslog	8.16.0-1ubuntu3
run-one	1.17-0ubuntu1
sbsigntool	0.6-0ubuntu10.1
screen	4.3.1-2build1
seabios	1.8.2-1ubuntu1
secureboot-db	1.1
sed	4.2.2-7
sensible-utils	0.0.9ubuntu0.16.04.1
sgml-base	1.26+nmu4ubuntu1
shared-mime-info	1.5-2ubuntu0.1
sharutils	1:4.15.2-1
shim	13-0ubuntu2
shim-signed	1.33.1~16.04.1+13-0ubuntu2

snap-confine	2.29.4.2
snapd	2.29.4.2
software-properties-common	0.96.20.7
sosreport	3.5-1~ubuntu16.04.2
squashfs-tools	1:4.3-3ubuntu2.16.04.1
ssh-import-id	5.5-0ubuntu1
strace	4.11-1ubuntu3
sudo	1.8.16-0ubuntu1.5
systemd	229-4ubuntu21.1
systemd-sysv	229-4ubuntu21.1
sysv-rc	2.88dsf-59.3ubuntu2
sysvinit-utils	2.88dsf-59.3ubuntu2
tar	1.28-2.1ubuntu0.1
tasksel	3.34ubuntu3
tasksel-data	3.34ubuntu3
tcpd	7.6.q-25
tcpdump	4.9.2-0ubuntu0.16.04.1
telnet	0.17-40
time	1.7-25.1
tmux	2.1-3build1
tzdata	2017c-0ubuntu0.16.04
ubuntu-cloudimage-keyring	2013.11.11
ubuntu-core-launcher	2.29.4.2
ubuntu-keyring	2012.05.19
ubuntu-minimal	1.361.1
ubuntu-release-upgrader-core	1:16.04.24
ubuntu-server	1.361.1
ubuntu-standard	1.361.1
ucf	3.0036
udev	229-4ubuntu21.1
ufw	0.35-0ubuntu2
uidmap	1:4.2-3.1ubuntu5.3
unattended-upgrades	0.90ubuntu0.9
unzip	6.0-20ubuntu1
update-manager-core	1:16.04.12
update-notifier-common	3.168.7
ureadahead	0.100.0-19
usbutils	1:007-4
util-linux	2.27.1-6ubuntu3.4
uuid-runtime	2.27.1-6ubuntu3.4
vbetool	1.1-3
vim	2:7.4.1689-3ubuntu1.2
vim-common	2:7.4.1689-3ubuntu1.2
vim-runtime	2:7.4.1689-3ubuntu1.2
vim-tiny	2:7.4.1689-3ubuntu1.2

vlan	1.9-3.2ubuntu1.16.04.4
wget	1.17.1-1ubuntu1.3
whiptail	0.52.18-1ubuntu2
wireless-regdb	2015.07.20-1ubuntu1
xauth	1:1.0.9-1ubuntu2
xdg-user-dirs	0.15-2ubuntu6
xfsprogs	4.3.0+nmu1ubuntu1.1
xkb-data	2.16-1ubuntu1
xml-core	0.13+nmu2
xz-utils	5.1.1alpha+20120614-2ubuntu2
zerofree	1.0.3-1
zlib1g:amd64	1:1.2.8.dfsg-2ubuntu4.1

APPENDIX B. CONFIGURATION FILES OF THE VIRTUALIZATION NODE

Content of the file `/etc/fstab`

```
# /etc/fstab: static file system information.
#
# Use 'blkid' to print the universally unique identifier for a
# device; this may be used with UUID= as a more robust way to name devices
# that works even if disks are added and removed. See fstab(5).
#
# <file system> <mount point>          <type> <options>          <dump> <pass>
/dev/sda2        /                ext4    errors=remount-ro 0       1
/dev/sda1        /boot/efi        vfat    umask=0077        0       1
/dev/sda3        none             swap    sw                0       0
/dev/sdb4        /mnt/forensic-data ext4    errors=remount-ro 0       2
```

Content of the file `/etc/network/interfaces`

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

auto enp7s0f0
iface enp7s0f0 inet manual

auto br0
iface br0 inet static
    address 192.168.10.1
    netmask 255.255.255.0
    network 192.168.10.0
    broadcast 192.168.10.255
    bridge_ports enp7s0f0
    bridge_stp off
    bridge_fd 0
    bridge_maxwait 0
```

Content of the file `/etc/libvirt/storage/lvm-group.xml`

```

<!--
WARNING: THIS IS AN AUTO-GENERATED FILE. CHANGES TO IT ARE LIKELY TO BE
OVERWRITTEN AND LOST. Changes to this xml configuration should be made
using: virsh pool-edit lvm-group or other application using the libvirt
API.
-->

```

```

<pool type='logical'>
  <name>lvm-group</name>
  <uuid>fb1197d6-d126-4ce9-a2ef-daf818092031</uuid>
  <capacity unit='bytes'>0</capacity>
  <allocation unit='bytes'>0</allocation>
  <available unit='bytes'>0</available>
  <source>
    <name>lvm-group</name>
    <format type='lvm2' />
  </source>
  <target>
    <path>/dev/lvm-group</path>
  </target>
</pool>

```

APPENDIX C. VIRTUAL MACHINES CONFIGURATION FILES

Content of the file /mnt/kvm/xmls/VM1.xml.

```
<domain type='kvm'>
  <name>VM1_Ubuntu_17.10</name>
  <uuid>be0c72a2-c9d7-4dd5-83e3-b2084387be81</uuid>
  <memory unit='KiB'>4194304</memory>
  <currentMemory unit='KiB'>4194304</currentMemory>
  <vcpu placement='static'>8</vcpu>
  <resource>
    <partition>/machine</partition>
  </resource>
  <os>
    <type arch='x86_64' machine='pc-i440fx-xenial'>hvm</type>
    <boot dev='hd' />
  </os>
  <features>
    <acpi />
    <apic />
  </features>
  <cpu mode='custom' match='exact'>
    <model fallback='allow'>Penryn</model>
  </cpu>
  <clock offset='utc'>
    <timer name='rtc' tickpolicy='catchup' />
    <timer name='pit' tickpolicy='delay' />
    <timer name='hpet' present='no' />
  </clock>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>restart</on_crash>
  <pm>
    <suspend-to-mem enabled='no' />
    <suspend-to-disk enabled='no' />
  </pm>
  <devices>
    <emulator>/usr/bin/kvm-spice</emulator>
    <disk type='block' device='disk'>
      <driver name='qemu' type='raw' cache='none' io='native' />
      <source dev='/dev/lvm-group/VM1_ubuntu' />
      <backingStore />
      <target dev='hda' bus='ide' />
      <alias name='ide0-0-0' />
    </disk>
  </devices>
</domain>
```

```

    <address type='drive' controller='0' bus='0' target='0' unit='0' />
</disk>
<controller type='usb' index='0' model='ich9-ehci1'>
    <alias name='usb' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x06' \
        function='0x7' />
</controller>
<controller type='usb' index='0' model='ich9-uhci1'>
    <alias name='usb' />
    <master startport='0' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x06' \
        function='0x0' multifunction='on' />
</controller>
<controller type='usb' index='0' model='ich9-uhci2'>
    <alias name='usb' />
    <master startport='2' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x06' \
        function='0x1' />
</controller>
<controller type='usb' index='0' model='ich9-uhci3'>
    <alias name='usb' />
    <master startport='4' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x06' \
        function='0x2' />
</controller>
<controller type='pci' index='0' model='pci-root'>
    <alias name='pci.0' />
</controller>
<controller type='ide' index='0'>
    <alias name='ide' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x01' \
        function='0x1' />
</controller>
<controller type='virtio-serial' index='0'>
    <alias name='virtio-serial0' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x05' \
        function='0x0' />
</controller>
<interface type='network'>
    <mac address='52:54:00:9e:d3:6c' />
    <source bridge='br0' />
    <target dev='vnet0' />
    <model type='rtl8139' />
    <alias name='net0' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x03' \

```

```

        function='0x0'>/>
</interface>
<serial type='pty'>
    <source path='/dev/pts/1'>/>
    <target port='0'>/>
    <alias name='serial0'>/>
</serial>
<console type='pty' tty='/dev/pts/1'>
    <source path='/dev/pts/1'>/>
    <target type='serial' port='0'>/>
    <alias name='serial0'>/>
</console>
<channel type='spicevmc'>
    <target type='virtio' name='com.redhat.spice.0' state='disconnected'>/>
    <alias name='channel0'>/>
    <address type='virtio-serial' controller='0' bus='0' port='1'>/>
</channel>
<input type='mouse' bus='ps2'>/>
<input type='keyboard' bus='ps2'>/>
<graphics type='spice' port='5900' autoport='yes' listen='127.0.0.1'>
    <listen type='address' address='127.0.0.1'>/>
</graphics>
<video>
    <model type='qxl' ram='65536' vram='65536' vgamem='16384' heads='1'>/>
    <alias name='video0'>/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x02' \
        function='0x0'>/>
</video>
<redirdev bus='usb' type='spicevmc'>
    <alias name='redir0'>/>
</redirdev>
<redirdev bus='usb' type='spicevmc'>
    <alias name='redir1'>/>
</redirdev>
<memballoon model='virtio'>
    <alias name='balloon0'>/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x07' \
        function='0x0'>/>
</memballoon>
</devices>
<seclabel type='dynamic' model='apparmor' relabel='yes'>
    <label>libvirt-be0c72a2-c9d7-4dd5-83e3-b2084387be81</label>
    <imagelabel>libvirt-be0c72a2-c9d7-4dd5-83e3-b2084387be81</imagelabel>
</seclabel>
</domain>

```

Content of the file /mnt/kvm/xmls/VM2.xml.

```
<domain type='kvm'>
  <name>VM2_CentOS_7</name>
  <uuid>999170f4-75c1-4d45-b875-fb77cc2a00ea</uuid>
  <memory unit='KiB'>4194304</memory>
  <currentMemory unit='KiB'>4194304</currentMemory>
  <vcpu placement='static'>8</vcpu>
  <resource>
    <partition>/machine</partition>
  </resource>
  <os>
    <type arch='x86_64' machine='pc-i440fx-xenial'>hvm</type>
    <boot dev='hd' />
  </os>
  <features>
    <acpi />
    <apic />
  </features>
  <cpu mode='custom' match='exact'>
    <model fallback='allow'>Penryn</model>
  </cpu>
  <clock offset='utc'>
    <timer name='rtc' tickpolicy='catchup' />
    <timer name='pit' tickpolicy='delay' />
    <timer name='hpet' present='no' />
  </clock>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>restart</on_crash>
  <pm>
    <suspend-to-mem enabled='no' />
    <suspend-to-disk enabled='no' />
  </pm>
  <devices>
    <emulator>/usr/bin/kvm-spice</emulator>
    <disk type='block' device='disk'>
      <driver name='qemu' type='raw' cache='none' io='native' />
      <source dev='/dev/lvm-group/VM2_centos' />
      <backingStore />
      <target dev='hda' bus='ide' />
      <alias name='ide0-0-0' />
      <address type='drive' controller='0' bus='0' target='0' unit='0' />
    </disk>
    <controller type='usb' index='0' model='ich9-ehci1'>
```

```

    <alias name='usb' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x05' \
function='0x7' />
</controller>
<controller type='usb' index='0' model='ich9-uhci1'>
    <alias name='usb' />
    <master startport='0' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x05' \
function='0x0' multifunction='on' />
</controller>
<controller type='usb' index='0' model='ich9-uhci2'>
    <alias name='usb' />
    <master startport='2' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x05' \
function='0x1' />
</controller>
<controller type='usb' index='0' model='ich9-uhci3'>
    <alias name='usb' />
    <master startport='4' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x05' \
function='0x2' />
</controller>
<controller type='pci' index='0' model='pci-root'>
    <alias name='pci.0' />
</controller>
<controller type='ide' index='0'>
    <alias name='ide' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x01' \
function='0x1' />
</controller>
<controller type='virtio-serial' index='0'>
    <alias name='virtio-serial0' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x04' \
function='0x0' />
</controller>
<interface type='bridge'>
    <mac address='52:54:00:f6:eb:05' />
    <source bridge='br0' />
    <target dev='vnet1' />
    <model type='rtl8139' />
    <alias name='net0' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x03' \
function='0x0' />
</interface>
<serial type='pty'>

```



```

    <source path='/dev/pts/5'/>
    <target port='0'/>
    <alias name='serial0'/>
</serial>
<console type='pty' tty='/dev/pts/5'>
    <source path='/dev/pts/5'/>
    <target type='serial' port='0'/>
    <alias name='serial0'/>
</console>
<channel type='spicevmc'>
    <target type='virtio' name='com.redhat.spice.0' state='disconnected'/>
    <alias name='channel0'/>
    <address type='virtio-serial' controller='0' bus='0' port='1'/>
</channel>
<input type='mouse' bus='ps2'/>
<input type='keyboard' bus='ps2'/>
<graphics type='spice' port='5901' autoport='yes' listen='127.0.0.1'>
    <listen type='address' address='127.0.0.1'/>
</graphics>
<video>
    <model type='qxl' ram='65536' vram='65536' vgamem='16384' heads='1'/>
    <alias name='video0'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x02' \
function='0x0'/>
</video>
<redirdev bus='usb' type='spicevmc'>
    <alias name='redir0'/>
</redirdev>
<redirdev bus='usb' type='spicevmc'>
    <alias name='redir1'/>
</redirdev>
<memballoon model='virtio'>
    <alias name='balloon0'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x06' \
function='0x0'/>
</memballoon>
</devices>
<seclabel type='dynamic' model='apparmor' relabel='yes'>
    <label>libvirt-999170f4-75c1-4d45-b875-fb77cc2a00ea</label>
    <imagelabel>libvirt-999170f4-75c1-4d45-b875-fb77cc2a00ea</imagelabel>
</seclabel>
</domain>

```

Content of the file /mnt/kvm/xmls/VM3.xml.

```

<domain type='kvm'>
  <name>VM3_Windows_2008</name>
  <uuid>9f44405d-2df8-483d-92f8-2050756235fd</uuid>
  <memory unit='KiB'>4194304</memory>
  <currentMemory unit='KiB'>4194304</currentMemory>
  <vcpu placement='static'>8</vcpu>
  <resource>
    <partition>/machine</partition>
  </resource>
  <os>
    <type arch='x86_64' machine='pc-i440fx-xenial'>hvm</type>
    <boot dev='hd' />
  </os>
  <features>
    <acpi />
    <apic />
    <hyperv>
      <relaxed state='on' />
      <vapic state='on' />
      <spinlocks state='on' retries='8191' />
    </hyperv>
  </features>
  <cpu mode='custom' match='exact'>
    <model fallback='allow'>Penryn</model>
  </cpu>
  <clock offset='localtime'>
    <timer name='rtc' tickpolicy='catchup' />
    <timer name='pit' tickpolicy='delay' />
    <timer name='hpet' present='no' />
    <timer name='hypervclock' present='yes' />
  </clock>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>restart</on_crash>
  <pm>
    <suspend-to-mem enabled='no' />
    <suspend-to-disk enabled='no' />
  </pm>
  <devices>
    <emulator>/usr/bin/kvm-spice</emulator>
    <disk type='block' device='disk'>
      <driver name='qemu' type='raw' cache='none' io='native' />
      <source dev='/dev/lvm-group/VM3_win2008' />
      <backingStore />
      <target dev='hda' bus='ide' />
    </disk>
  </devices>
</domain>

```

```

    <alias name='ide0-0-0' />
    <address type='drive' controller='0' bus='0' target='0' unit='0' />
</disk>
<controller type='usb' index='0' model='ich9-ehci1'>
    <alias name='usb' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x05' \
    function='0x7' />
</controller>
<controller type='usb' index='0' model='ich9-uhci1'>
    <alias name='usb' />
    <master startport='0' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x05' \
    function='0x0' multifunction='on' />
</controller>
<controller type='usb' index='0' model='ich9-uhci2'>
    <alias name='usb' />
    <master startport='2' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x05' \
    function='0x1' />
</controller>
<controller type='usb' index='0' model='ich9-uhci3'>
    <alias name='usb' />
    <master startport='4' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x05' \
    function='0x2' />
</controller>
<controller type='pci' index='0' model='pci-root'>
    <alias name='pci.0' />
</controller>
<controller type='ide' index='0'>
    <alias name='ide' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x01' \
    function='0x1' />
</controller>
<controller type='virtio-serial' index='0'>
    <alias name='virtio-serial0' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x04' \
    function='0x0' />
</controller>
<interface type='network'>
    <mac address='52:54:00:70:30:7f' />
    <source bridge='br0' />
    <target dev='vnet2' />
    <model type='rtl8139' />
    <alias name='net0' />

```

```

        <address type='pci' domain='0x0000' bus='0x00' slot='0x03' \
        function='0x0' />
    </interface>
    <serial type='pty'>
        <source path='/dev/pts/6' />
        <target port='0' />
        <alias name='serial0' />
    </serial>
    <console type='pty' tty='/dev/pts/6'>
        <source path='/dev/pts/6' />
        <target type='serial' port='0' />
        <alias name='serial0' />
    </console>
    <channel type='spicevmc'>
        <target type='virtio' name='com.redhat.spice.0' state='disconnected' />
        <alias name='channel0' />
        <address type='virtio-serial' controller='0' bus='0' port='1' />
    </channel>
    <input type='tablet' bus='usb'>
        <alias name='input0' />
    </input>
    <input type='mouse' bus='ps2' />
    <input type='keyboard' bus='ps2' />
    <graphics type='spice' port='5902' autoport='yes' listen='127.0.0.1'>
        <listen type='address' address='127.0.0.1' />
    </graphics>
    <video>
        <model type='qxl' ram='65536' vram='65536' vgamem='16384' heads='1' />
        <alias name='video0' />
        <address type='pci' domain='0x0000' bus='0x00' slot='0x02' \
        function='0x0' />
    </video>
    <redirdev bus='usb' type='spicevmc'>
        <alias name='redir0' />
    </redirdev>
    <redirdev bus='usb' type='spicevmc'>
        <alias name='redir1' />
    </redirdev>
    <memballoon model='virtio'>
        <alias name='balloon0' />
        <address type='pci' domain='0x0000' bus='0x00' slot='0x06' \
        function='0x0' />
    </memballoon>
</devices>
<seclabel type='dynamic' model='apparmor' relabel='yes'>

```

```

    <label>libvirt-9f44405d-2df8-483d-92f8-2050756235fd</label>
    <imagelabel>libvirt-9f44405d-2df8-483d-92f8-2050756235fd</imagelabel>
  </seclabel>
</domain>

```

Content of the file /mnt/kvm/xmls/VM4.xml.

```

<domain type='kvm'>
  <name>VM4_Windows_2016</name>
  <uuid>5463353b-7bf4-49f0-b48f-1ff20d9c499a</uuid>
  <memory unit='KiB'>4194304</memory>
  <currentMemory unit='KiB'>4194304</currentMemory>
  <vcpu placement='static'>8</vcpu>
  <resource>
    <partition>/machine</partition>
  </resource>
  <os>
    <type arch='x86_64' machine='pc-i440fx-xenial'>hvm</type>
  </os>
  <features>
    <acpi/>
    <apic/>
  </features>
  <cpu mode='custom' match='exact'>
    <model fallback='allow'>Penryn</model>
  </cpu>
  <clock offset='utc'>
    <timer name='rtc' tickpolicy='catchup'/>
    <timer name='pit' tickpolicy='delay'/>
    <timer name='hpet' present='no'/>
  </clock>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>restart</on_crash>
  <pm>
    <suspend-to-mem enabled='no'/>
    <suspend-to-disk enabled='no'/>
  </pm>
  <devices>
    <emulator>/usr/bin/kvm-spice</emulator>
    <disk type='block' device='disk'>
      <driver name='qemu' type='raw' cache='none' io='native'/>
      <source dev='/dev/lvm-group/VM4_win2016'/>
      <backingStore/>
      <target dev='hda' bus='ide'/>
    </disk>
  </devices>
</domain>

```

```

    <boot order='2' />
    <alias name='ide0-0-0' />
    <address type='drive' controller='0' bus='0' target='0' unit='0' />
</disk>
<controller type='usb' index='0' model='ich9-ehci1'>
    <alias name='usb' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x05' \
    function='0x7' />
</controller>
<controller type='usb' index='0' model='ich9-uhci1'>
    <alias name='usb' />
    <master startport='0' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x05' \
    function='0x0' multifunction='on' />
</controller>
<controller type='usb' index='0' model='ich9-uhci2'>
    <alias name='usb' />
    <master startport='2' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x05' \
    function='0x1' />
</controller>
<controller type='usb' index='0' model='ich9-uhci3'>
    <alias name='usb' />
    <master startport='4' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x05' \
    function='0x2' />
</controller>
<controller type='pci' index='0' model='pci-root'>
    <alias name='pci.0' />
</controller>
<controller type='ide' index='0'>
    <alias name='ide' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x01' \
    function='0x1' />
</controller>
<controller type='virtio-serial' index='0'>
    <alias name='virtio-serial0' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x04' \
    function='0x0' />
</controller>
<interface type='network'>
    <mac address='52:54:00:24:13:f5' />
    <source bridge='br0' />
    <target dev='vnet3' />
    <model type='rtl8139' />

```

```

    <alias name='net0' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x03' \
    function='0x0' />
</interface>
<serial type='pty'>
    <source path='/dev/pts/2' />
    <target port='0' />
    <alias name='serial0' />
</serial>
<console type='pty' tty='/dev/pts/2'>
    <source path='/dev/pts/2' />
    <target type='serial' port='0' />
    <alias name='serial0' />
</console>
<channel type='spicevmc'>
    <target type='virtio' name='com.redhat.spice.0' state='disconnected' />
    <alias name='channel0' />
    <address type='virtio-serial' controller='0' bus='0' port='1' />
</channel>
<input type='mouse' bus='ps2' />
<input type='keyboard' bus='ps2' />
<graphics type='spice' port='5903' autoport='yes' listen='127.0.0.1'>
    <listen type='address' address='127.0.0.1' />
</graphics>
<video>
    <model type='qxl' ram='65536' vram='65536' vgamem='16384' heads='1' />
    <alias name='video0' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x02' \
    function='0x0' />
</video>
<redirdev bus='usb' type='spicevmc'>
    <alias name='redir0' />
</redirdev>
<redirdev bus='usb' type='spicevmc'>
    <alias name='redir1' />
</redirdev>
<memballoon model='virtio'>
    <alias name='balloon0' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x06' \
    function='0x0' />
</memballoon>
</devices>
<seclabel type='dynamic' model='apparmor' relabel='yes'>
    <label>libvirt-5463353b-7bf4-49f0-b48f-1ff20d9c499a</label>
    <imagelabel>libvirt-5463353b-7bf4-49f0-b48f-1ff20d9c499a</imagelabel>

```

```
</seclabel>  
</domain>
```


APPENDIX D. GENERATION OF VOLATILITY PROFILES

Profile generation steps for VM1 (Ubuntu Server 17.10 32-bit):

```
# apt-get install vim mc zip unzip linux-headers-generic libelf-dev
# build-essential
# cd /root
# wget http://downloads.volatilityfoundation.org/releases/2.6/
# volatility-2.6.zip
# unzip volatility-2.6.zip
# git clone https://github.com/tomhughes/libdwarf.git
# cd libdwarf/
# ./configure
# make dd
# cp dwarfdump/dwarfdump /usr/local/bin/
# cp dwarfdump/dwarfdump.conf /usr/local/lib/
# cp libdwarf/libdwarf.a /usr/local/lib
# cd /root/volatility-master/tools/linux/
# make
# zip /root/Ubuntu1710.zip /root/volatility-master/tools/linux/module.dwarf
# /boot/System.map-4.13.0-36-generic
```

The file /root/Ubuntu1710.zip generated in the last step is the Volatility profile which was used to examine the RAM images created for VM1.

Profile generation steps for VM2 (CentOS 7 64-bit):

```
# yum install vim mc telnet net-tools mlocate wget git gcc zip
# unzip elfutils libdwarf libdwarf-devel elfutils-libelf-devel
# kernel-devel kernel-headers
# cd /root
# wget http://downloads.volatilityfoundation.org/releases/2.6/
# volatility-2.6.zip
# unzip volatility-2.6.zip
# git clone https://github.com/tomhughes/libdwarf.git
# cd libdwarf/
# ./configure
# make dd
# cp dwarfdump/dwarfdump /usr/local/bin/
# cp dwarfdump/dwarfdump.conf /usr/local/lib/
# cp libdwarf/libdwarf.a /usr/local/lib
# cd /root/volatility-master/tools/linux/
# make
# zip /root/Centos7.zip /root/volatility-master/tools/linux/module.dwarf
# /boot/System.map-3.10.0-693.21.1.el7.x86_64
```

The file `/root/Centos7.zip` generated in the last step is the Volatility profile which was used to examine the RAM images created for VM2.

APPENDIX E. BASH SCRIPT TO TEST THE IMAGING PROCESS

BASH script developed to test the imaging process of virtual hard drives by employing the utility virsh to create a snapshot.

```
#!/bin/bash

VM_NAME=('VM1_Ubuntu_17.10' 'VM2_CentOS_7' 'VM3_Windows_2008'
        'VM4_Windows_2016')
VM_LVM=('/dev/lvm-group/VM1_ubuntu' '/dev/lvm-group/VM2_centos'
        '/dev/lvm-group/VM3_win2008' '/dev/lvm-group/VM4_win2016')
VM_MAPPER=('/dev/mapper/loop0p1' '/dev/mapper/loop0p1'
           '/dev/mapper/loop0p1' '/dev/mapper/loop0p2')
VM_FILE=('/var/log/syslog' '/var/log/messages'
        '/Windows/Logs/ServerManager.log' '/Windows/debug/wlms.log')
VM_DISK_DEV=('hda' 'hda' 'hda' 'hda')
VM_FS=('ext4' 'xfs' 'ntfs' 'ntfs')

RM=$(which rm)
DD=$(which dd)
MD5=$(which md5sum)
SHA1=$(which sha1sum)
TAIL=$(which tail)
VIRSH=$(which virsh)
MOUNT=$(which mount)
UMOUNT=$(which umount)
KPARTX=$(which kpartx)
SNAP_DIR="/var/lib/libvirt/images"
SNAP_NAME="snapshot-script"
IMAGE_DIR="/mnt/forensic-data/disk-images"
IMAGE_DEV="/dev/sdb2"
MOUNT_TMPDIR="/mnt/temp"

for i in {0..3}; do
    echo ----- ${VM_NAME[i]} -----
    # Step 1: Create the external snapshot
    echo -n "STEP 1: CREATING EXTERNAL SNAPSHOT. Date: "
    date +"%m-%d-%y %I:%M:%S %p"
    /usr/bin/time -f "Time: %E" $VIRSH snapshot-create-as ${VM_NAME[i]} \
    --disk-only --atomic --name $SNAP_NAME --diskspec ${VM_DISK_DEV[i]},\
    snapshot=external,file=$SNAP_DIR/${VM_NAME[i]}-snapshot.qcow2
    if [ $? -eq 0 ]; then
        echo "Result: Snapshot successfully created."
    else
```

```

    echo "Result: ERROR! Snapshot could not be created."
    exit
fi
echo

# Step 2: Create bit-stream image and calculate hash values
echo -n "STEP 2: CREATING IMAGE AND CALCULATING HASH VALUES. Date: "
date +"%m-%d-%y %I:%M:%S %p"
echo " * Creating bit-stream image into /mnt/forensic-data:"
$DD if=${VM_LVM[i]} of=$IMAGE_DIR/${VM_NAME[i]}.dd bs=512 \
conv=noerror,sync,fdatasync
sleep 5m
echo " * Creating bit-stream image into /dev/sdb2:"
$DD if=${VM_LVM[i]} of=$IMAGE_DEV bs=512 conv=noerror,sync,fdatasync
echo " * Calculating MD5 values:"
$MD5 ${VM_LVM[i]}
$MD5 $IMAGE_DIR/${VM_NAME[i]}.dd
$MD5 $IMAGE_DEV
echo " * Calculating SHA1 values:"
$SHA1 ${VM_LVM[i]}
$SHA1 $IMAGE_DIR/${VM_NAME[i]}.dd
$SHA1 $IMAGE_DEV
echo "Result: TBD (Check MD5 and SHA1 values)."
echo

# Step 3: Mounting bit-stream image
echo -n "STEP 3: VERIFYING IMAGE. Date: "
date +"%m-%d-%y %I:%M:%S %p"

echo " * Mounting image:"
$KPARTX -av $IMAGE_DIR/${VM_NAME[i]}.dd ; sleep 3s
$MOUNT -t ${VM_FS[i]} -o ro ${VM_MAPPER[i]} $MOUNT_TMPDIR
if [ $? -eq 0 ]; then
    echo " * Retrieving file:"
    $TAIL -n2 $MOUNT_TMPDIR${VM_FILE[i]}
    $UMOUNT $MOUNT_TMPDIR ; sleep 3s
    $KPARTX -dv $IMAGE_DIR/${VM_NAME[i]}.dd
    echo "Result: TBD (Check date and time)."
else
    echo "Result: ERROR! Image could not be mounted."
    exit
fi
echo

# Step 4: Merge the snapshot into and remove files:

```

```

echo -n "STEP 4: MERGING SNAPSHOT AND REMOVING FILES. Date: "
date +"%m-%d-%y %I:%M:%S %p"

/usr/bin/time -f "Time: %E" $VIRSH blockcommit ${VM_NAME[i]} \
${VM_DISK_DEV[i]} --verbose --pivot --active
if [ $? -eq 0 ]; then
    echo "Result: Snapshot successfully merged."
    $VIRSH snapshot-delete ${VM_NAME[i]} $SNAP_NAME --metadata
    $RM $SNAP_DIR/${VM_NAME[i]}-snapshot.qcow2
    $RM $IMAGE_DIR/${VM_NAME[i]}.dd
else
    echo "Result: ERROR! Snapshot could not be merged."
fi
echo
echo "PROCESS COMPLETE. Date: "
date +"%m-%d-%y %I:%M:%S %p"
echo
sleep 15m
done

exit 0

```

APPENDIX F. PURDUE'S IRB EXEMPTION LETTER

Figure F.1 shows the review exemption letter provided by Purdue's Institutional Review Board (IRB) for the research conducted in this study.

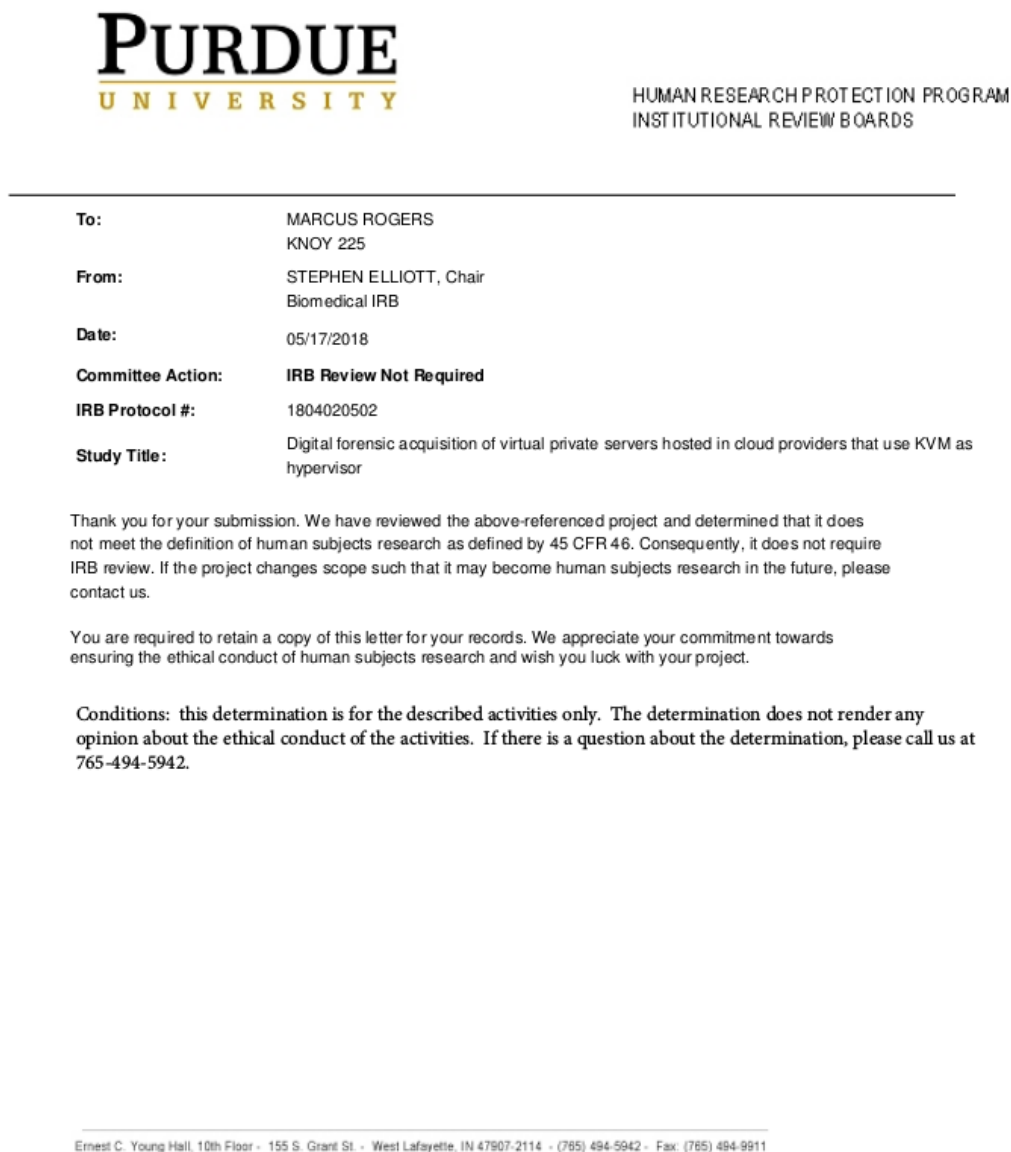
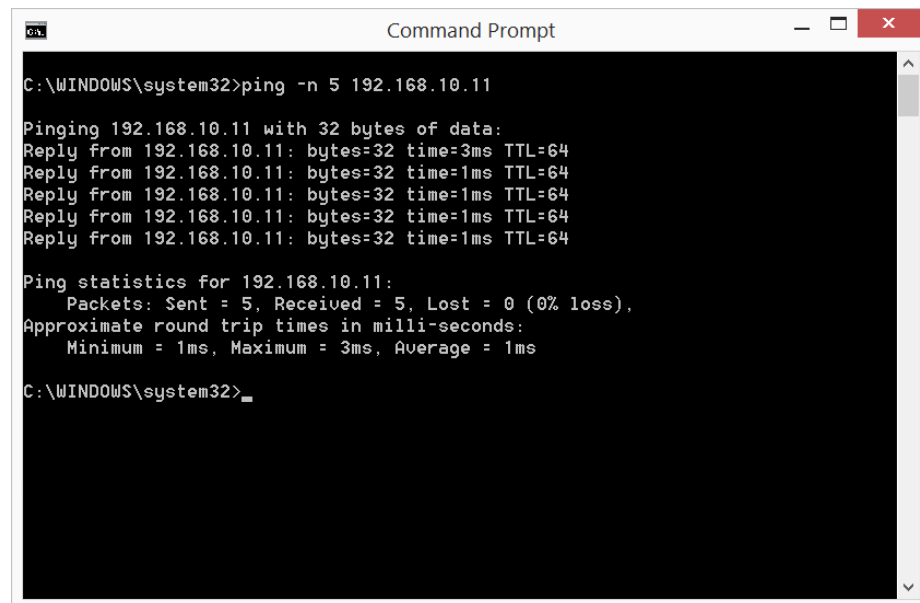


Figure F.1. Purdue's Institutional Review Board exemption letter

APPENDIX G. USER INTERACTION SCRIPT FOR VM1 (UBUNTU 17.10)

This document describes the steps the user must follow in order to interact with the VM Ubuntu 17.10 (IP address: 192.168.10.11):

1. Log in the laptop with the following credentials:
 - User: cflstudent
 - Password: 2hLMmVGt
2. Send an ICMP echo request to the VM:
 - 2.1. Open a command prompt window by double-clicking on the icon “Command Prompt”, located at the Desktop.
 - 2.2. Execute the command `ping -n X 192.168.10.11`, replacing X by a number between 1 and 10. Figure G.1 displays an example of this step using a value of 5 for X.



```
Command Prompt
C:\WINDOWS\system32>ping -n 5 192.168.10.11

Pinging 192.168.10.11 with 32 bytes of data:
Reply from 192.168.10.11: bytes=32 time=3ms TTL=64
Reply from 192.168.10.11: bytes=32 time=1ms TTL=64
Reply from 192.168.10.11: bytes=32 time=1ms TTL=64
Reply from 192.168.10.11: bytes=32 time=1ms TTL=64
Reply from 192.168.10.11: bytes=32 time=1ms TTL=64

Ping statistics for 192.168.10.11:
    Packets: Sent = 5, Received = 5, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 1ms, Maximum = 3ms, Average = 1ms

C:\WINDOWS\system32>
```

Figure G.1. Ping command screen

- 2.3. Close the command prompt window.
3. Connect to the SSH server running on the VM:
 - 3.1. Open the SSH client by double-clicking on the icon “PuTTY (64 bits)”, located at the Desktop.
 - 3.2. Double-click on the session “VM1: Ubuntu 17.10” that was previously created to connect to the IP address 192.168.10.11 and port TCP 22. Figure G.2 displays the Putty screen and the saved session list.

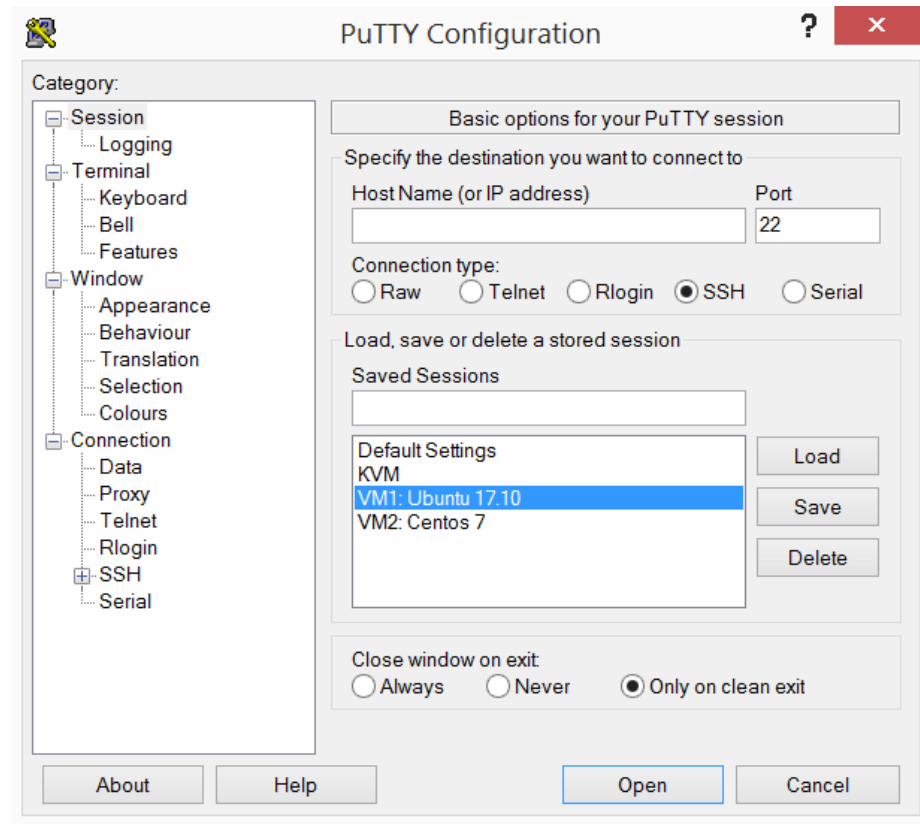


Figure G.2. Putty screen

3.3. Log in the Ubuntu system with the following credentials:

- User: cflstudent
- Password: 2hLMmVGt

3.4. Select two commands from the next list and execute them, once at a time:

date	ifconfig	ps	clear	df
ls	ip ro	who	w	whoami
last	cd	hostname	blkid	uname

3.5. Leave the SSH session open (do not close the terminal).

4. Connect to the webserver running on the VM:

- 4.1. Open the Google Chrome browser by double-clicking on the icon “Google Chrome”, located at the Desktop.
- 4.2. In the address bar enter the following URL: `http://192.168.10.11`
- 4.3. You should see a web page with the title “Apache2 Ubuntu Default Page”. Figure G.3 shows an example of the web page retrieved after entering the mentioned URL.

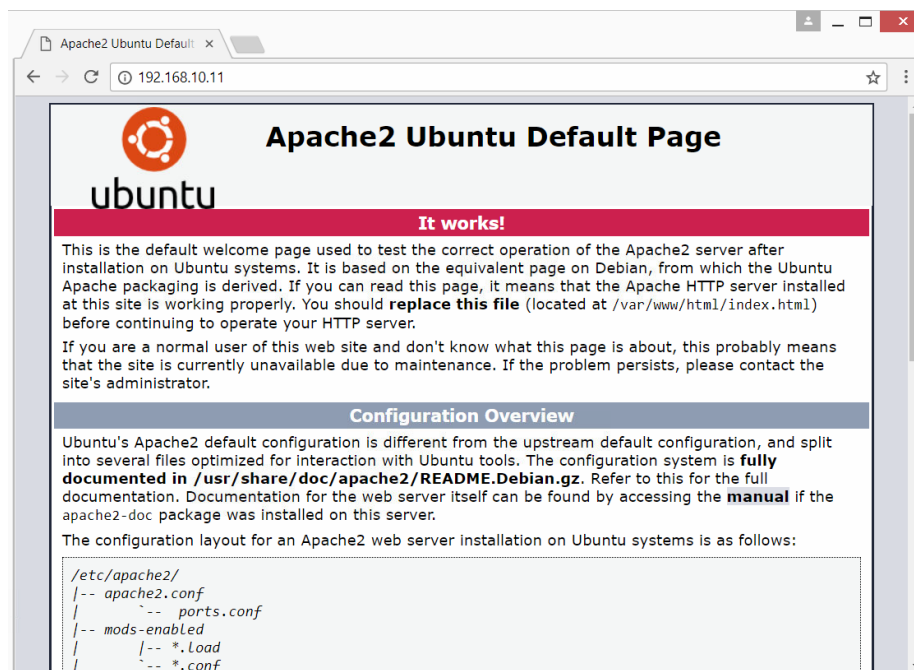


Figure G.3. Chrome screen

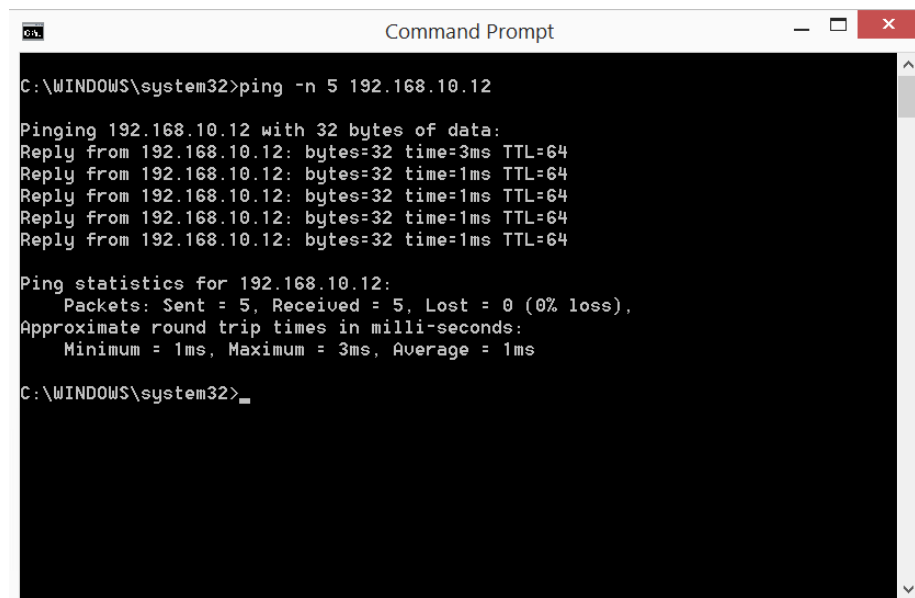
- 4.4. Close the Google Chrome browser.
5. Complete the first available row in the next table, detailing the date and time the interaction was performed, the X value selected at step 2.2 and the commands executed at step 3.4.

Test	Date	Time	X value	Commands executed
1	05/18	5:40pm	5	ip ro — uname
2	05/20	6:55pm	7	hostname — df
3	05/21	6:25pm	1	ps — clear
4	05/22	5:50pm	3	w — last
5	05/23	5:40pm	10	date — cd
6	05/25	4:10pm	8	df — hostname
7	05/30	6:05pm	3	date — ifconfig
8	06/01	7:15pm	5	clear — cd
9	06/05	11:10am	8	last — df
10	06/08	10:45am	9	whoami — blkid

APPENDIX H. USER INTERACTION SCRIPT FOR VM2 (CENTOS7)

This document describes the steps the user must follow in order to interact with the VM Centos 7 (IP address: 192.168.10.12):

1. Log in the laptop with the following credentials:
 - User: cflstudent
 - Password: 2hLMmVGt
2. Send an ICMP echo request to the VM:
 - 2.1. Open a command prompt window by double-clicking on the icon “Command Prompt”, located at the Desktop.
 - 2.2. Execute the command *ping -n X 192.168.10.12*, replacing X by a number between 1 and 10. Figure H.1 displays an example of this step using a value of 5 for X.



```
C:\WINDOWS\system32>ping -n 5 192.168.10.12

Pinging 192.168.10.12 with 32 bytes of data:
Reply from 192.168.10.12: bytes=32 time=3ms TTL=64
Reply from 192.168.10.12: bytes=32 time=1ms TTL=64
Reply from 192.168.10.12: bytes=32 time=1ms TTL=64
Reply from 192.168.10.12: bytes=32 time=1ms TTL=64
Reply from 192.168.10.12: bytes=32 time=1ms TTL=64

Ping statistics for 192.168.10.12:
    Packets: Sent = 5, Received = 5, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 1ms, Maximum = 3ms, Average = 1ms

C:\WINDOWS\system32>
```

Figure H.1. Ping command screen

- 2.3. Close the command prompt window.
3. Connect to the SSH server running on the VM:
 - 3.1. Open the SSH client by double-clicking on the icon “PuTTY (64 bits)”, located at the Desktop.
 - 3.2. Double-click on the session “VM2: Centos 7” that was previously created to connect to the IP address 192.168.10.12 and port TCP 22. Figure H.2 displays the Putty screen and the saved session list.

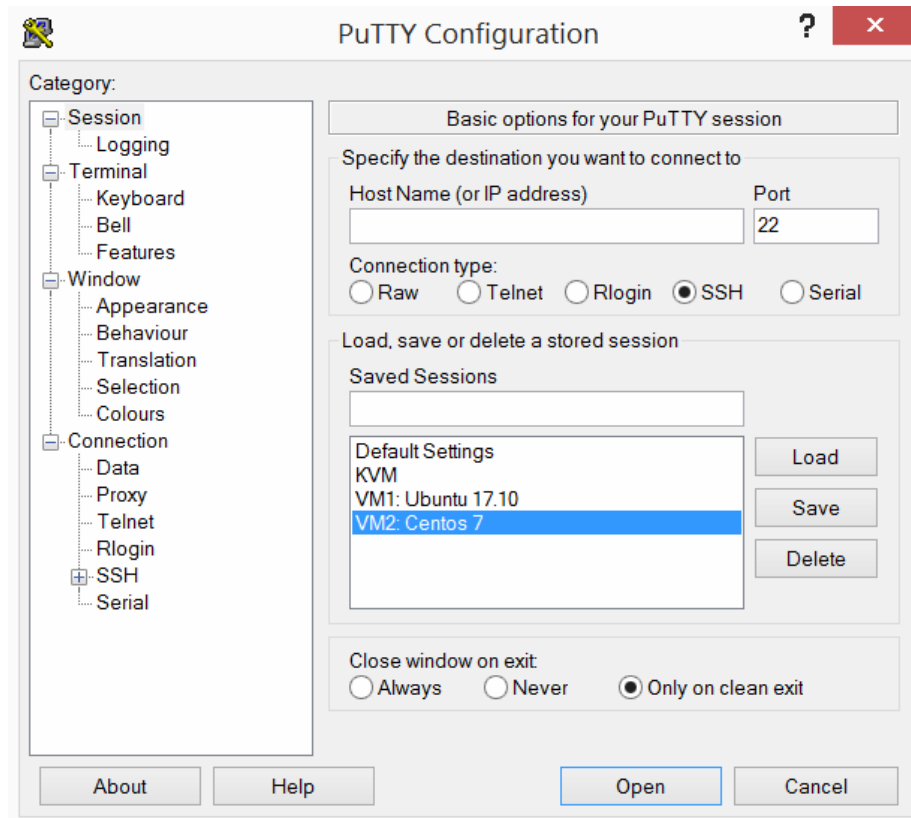


Figure H.2. Putty screen

3.3. Log in the Centos system with the following credentials:

- User: cflstudent
- Password: 2hLMmVGt

3.4. Select two commands from the next list and execute them, once at a time:

date	ifconfig	ps	clear	df
ls	ip ro	who	w	whoami
last	cd	hostname	blkid	uname

3.5. Leave the SSH session open (do not close the terminal).

4. Connect to the FTP server running on the VM:

- 4.1. Open the FileZilla FTP Client by double-clicking on the icon “FileZilla Client”, located at the Desktop.
- 4.2. Go to menu “File” and then select “Site Manager...”.
- 4.3. Double-click on the entry “Centos 7” that was previously created to connect to the IP address 192.168.10.12 and port TCP 21, using the same credentials

mentioned in the previous step. Figure H.3 displays the FileZilla Site Manager screen and the aforementioned entry.

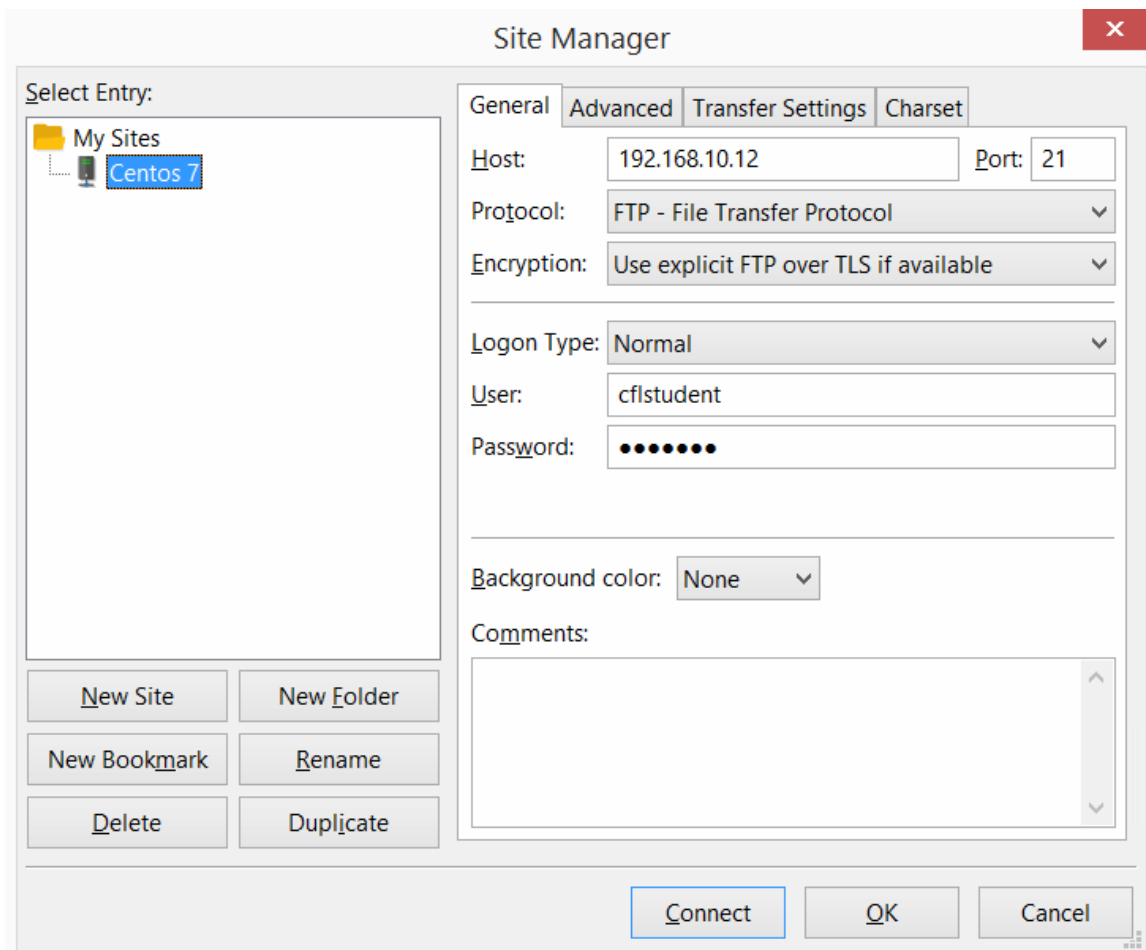


Figure H.3. FileZilla Site Manager screen

- 4.4. Wait until the FTP connection is established. Figure H.4 illustrates the FileZilla screen at this point, after the content of the directory /home/cflstudent is listed.

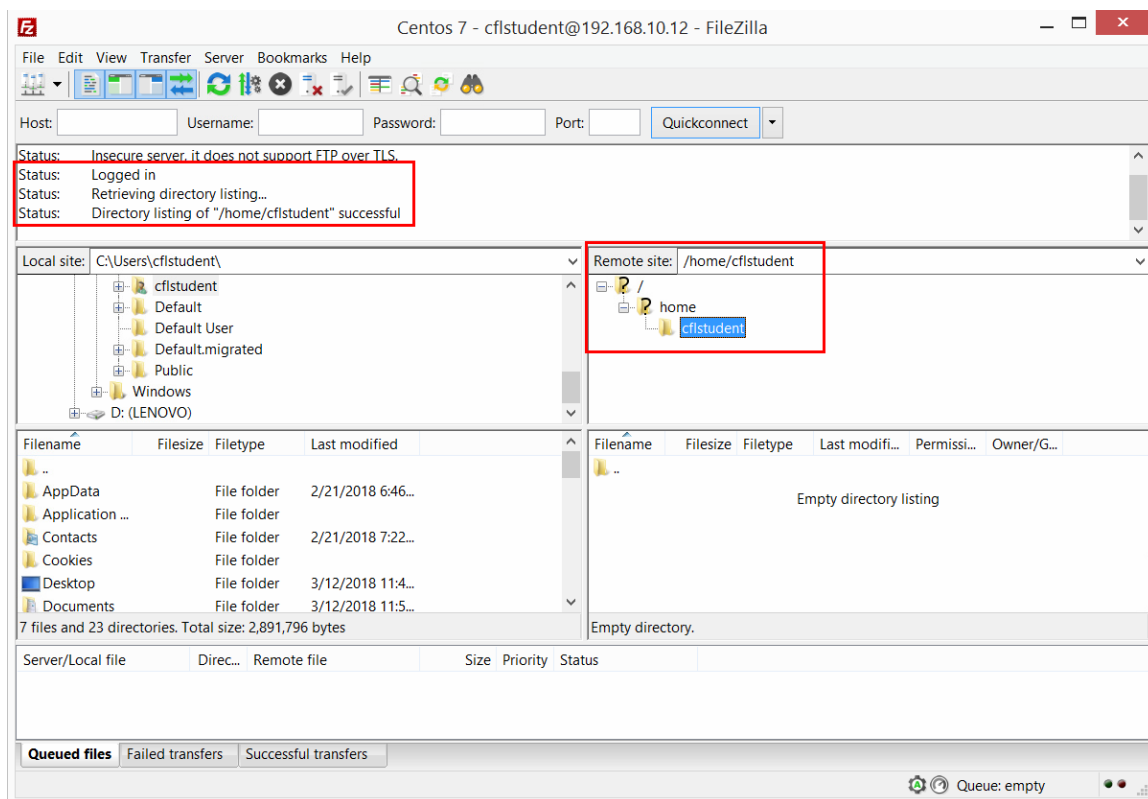


Figure H.4. FileZilla screen once the FTP connection is established

4.5. Close the FileZilla FTP Client.

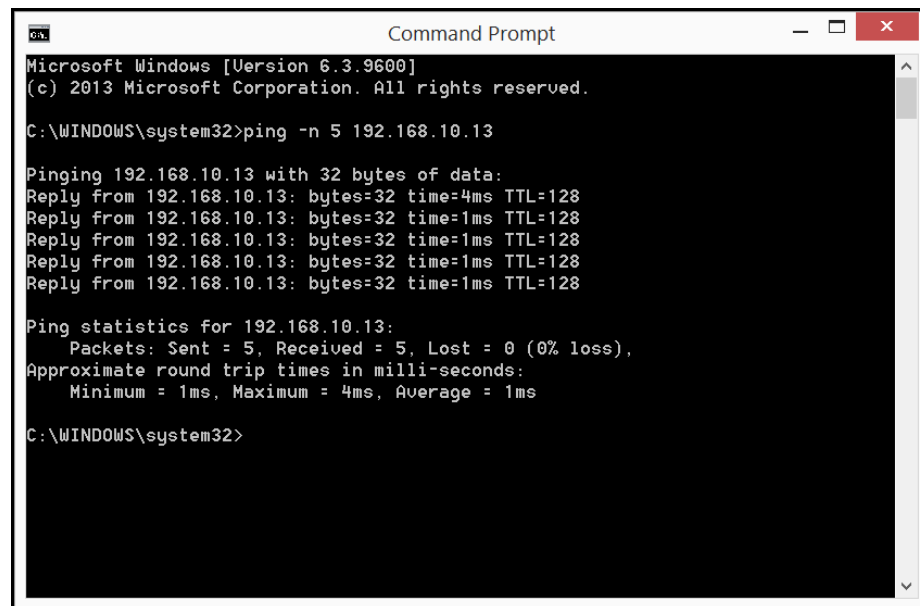
- Complete the first available row in the next table, detailing the date and time the interaction was performed, the X value selected at step 2.2 and the commands executed at step 3.4.

Test	Date	Time	X value	Commands executed
1	05/18	5:50pm	10	blkid — whoami
2	05/20	7:10pm	9	ifconfig — date
3	05/21	6:30pm	4	ip ro — who
4	05/22	6:00pm	7	ifconfig — uname
5	05/24	7:00pm	1	last — whoami
6	05/25	4:20pm	4	ip ro — blkid
7	05/30	6:20pm	5	ps — uname
8	06/01	7:20pm	10	ifconfig — last
9	06/05	11:15am	2	hostname — w
10	06/08	10:50am	9	ip ro — last

APPENDIX I. USER INTERACTION SCRIPT FOR VM3 (WINDOWS 2008)

This document describes the steps the user must follow in order to interact with the VM Windows 2008 (IP address: 192.168.10.13):

1. Log in the laptop with the following credentials:
 - User: cflstudent
 - Password: 2hLMmVGt
2. Send an ICMP echo request to the VM:
 - 2.1. Open a command prompt window by double-clicking on the icon “Command Prompt”, located at the Desktop.
 - 2.2. Execute the command `ping -n X 192.168.10.13`, replacing X by a number between 1 and 10. Figure I.1 displays an example of this step using a value of 5 for X.



```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>ping -n 5 192.168.10.13

Pinging 192.168.10.13 with 32 bytes of data:
Reply from 192.168.10.13: bytes=32 time=4ms TTL=128
Reply from 192.168.10.13: bytes=32 time=1ms TTL=128
Reply from 192.168.10.13: bytes=32 time=1ms TTL=128
Reply from 192.168.10.13: bytes=32 time=1ms TTL=128
Reply from 192.168.10.13: bytes=32 time=1ms TTL=128

Ping statistics for 192.168.10.13:
    Packets: Sent = 5, Received = 5, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 1ms, Maximum = 4ms, Average = 1ms

C:\WINDOWS\system32>
```

Figure I.1. Ping command screen

- 2.3. Close the command prompt window.
3. Connect to the Remote Desktop service running on the VM:
 - 3.1. Open the remote desktop client by double-clicking on the icon “Remote Desktop”, located at the Desktop.
 - 3.2. In the Computer text-box write the IP address 192.168.10.13 and then click on the button “Connect”. Figure I.2 displays the remote desktop client screen.

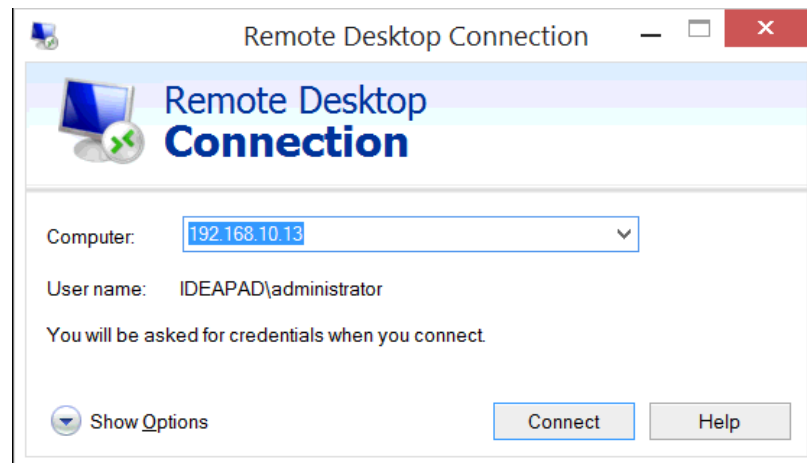


Figure I.2. Remote desktop client screen

3.3. Log in the Windows 2008 system with the following credentials:

- User: Administrator
- Password: 2hLMmVGt

3.4. After log in the Windows 2008, complete the next actions inside this system:

- 3.4.1. Open a command prompt window by double-clicking on the icon “Command Prompt”, located at the Windows 2008 Desktop.
- 3.4.2. Select two commands from the next list and execute them, inside the command prompt window, once at a time:

cls	ipconfig	route print	dir	cd
date /t	hostname	netstat	tree	ver
time /t	vol	whoami	arp -a	dispdiag

3.4.3. Minimize the command prompt window. Do not close it. If it was closed by mistake start again from step 3.4.1.

3.5. Once the previous actions are completed, leave the remote desktop connection open (do not close the session, just minimize it to continue with the next step).

4. Query the DNS server running on the VM:

- 4.1. Open a command prompt window by double-clicking on the icon “Command Prompt”, located at the Desktop.
- 4.2. Execute the command `nslookup.exe XXX.vpsnet.com 192.168.10.13`, replacing XXX by one of the following options: vm1, vm2, vm3, or vm4. Depending on the option selected, the IP address reported by the DNS server could be: 192.168.10.11 (for vm1), 192.168.10.12 (for vm2), 192.168.10.13 (for vm3), or 192.168.10.14 (for vm4). Figure I.3 displays an example of this step using the

value “vm1” for XXX. In this example, the IP address reported by the DNS server was 192.168.10.11.

```

Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>nslookup.exe vm1.upsnet.com 192.168.10.13
DNS request timed out.
    timeout was 2 seconds.
Server: UnKnown
Address: 192.168.10.13

DNS request timed out.
    timeout was 2 seconds.
DNS request timed out.
    timeout was 2 seconds.
Name:    vm1.upsnet.com
Address: 192.168.10.11

C:\WINDOWS\system32>

```

Figure I.3. Nslookup command screen

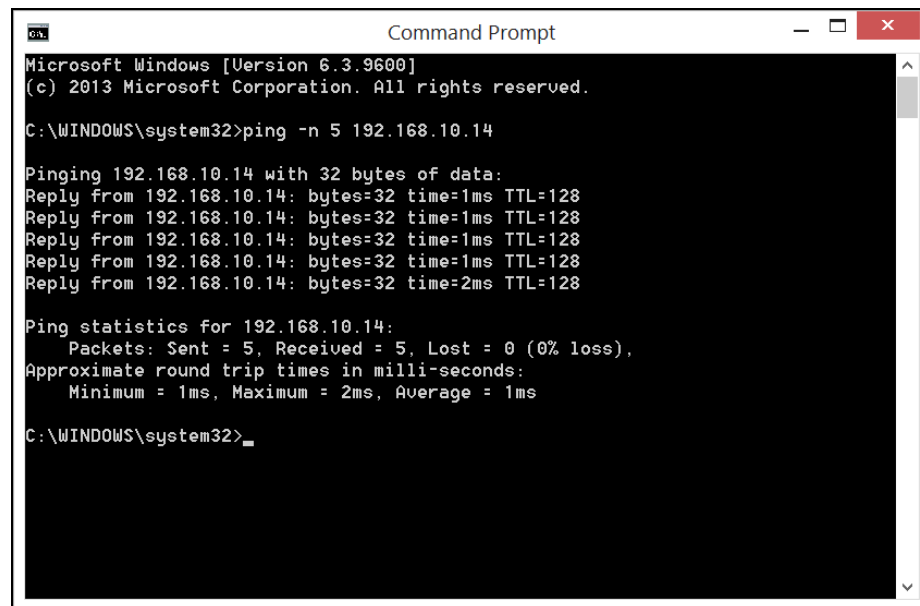
- 4.3. Close the command prompt window.
5. Complete the first available row in the next table, detailing the date and time the interaction was performed, the X value selected at step 2.2 and the commands executed at step 3.4.2.

Test	Date	Time	X value	Commands executed
1	05/18	5:55pm	8	route print — dispdiag
2	05/20	7:20pm	3	vol — ver
3	05/21	6:40pm	2	cls — netstat
4	05/22	6:30pm	5	time /t — date /t
5	05/23	6:10pm	6	tree — vol
6	05/25	4:25pm	1	cls — vol
7	05/30	6:25pm	4	ver — vol
8	06/01	7:25pm	8	ipconfig — arp -a
9	06/05	11:25am	9	date /t — netstat
10	06/08	10:54am	9	cls — date /t

APPENDIX J. USER INTERACTION SCRIPT FOR VM4 (WINDOWS 2016)

This document describes the steps the user must follow in order to interact with the VM Windows 2016 (IP address: 192.168.10.14):

1. Log in the laptop with the following credentials:
 - User: cflstudent
 - Password: 2hLMmVGt
2. Send an ICMP echo request to the VM:
 - 2.1. Open a command prompt window by double-clicking on the icon “Command Prompt”, located at the Desktop.
 - 2.2. Execute the command `ping -n X 192.168.10.14`, replacing X by a number between 1 and 10. Figure J.1 displays an example of this step using a value of 5 for X.



```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>ping -n 5 192.168.10.14

Pinging 192.168.10.14 with 32 bytes of data:
Reply from 192.168.10.14: bytes=32 time=1ms TTL=128
Reply from 192.168.10.14: bytes=32 time=1ms TTL=128
Reply from 192.168.10.14: bytes=32 time=1ms TTL=128
Reply from 192.168.10.14: bytes=32 time=1ms TTL=128
Reply from 192.168.10.14: bytes=32 time=2ms TTL=128

Ping statistics for 192.168.10.14:
    Packets: Sent = 5, Received = 5, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 1ms, Maximum = 2ms, Average = 1ms

C:\WINDOWS\system32>
```

Figure J.1. Ping command screen

- 2.3. Close the command prompt window.
3. Connect to the Remote Desktop service running on the VM:
 - 3.1. Open the remote desktop client by double-clicking on the icon “Remote Desktop”, located at the Desktop.
 - 3.2. In the Computer text-box write the IP address 192.168.10.14 and then click on the button “Connect”. Figure J.2 displays the remote desktop client screen.

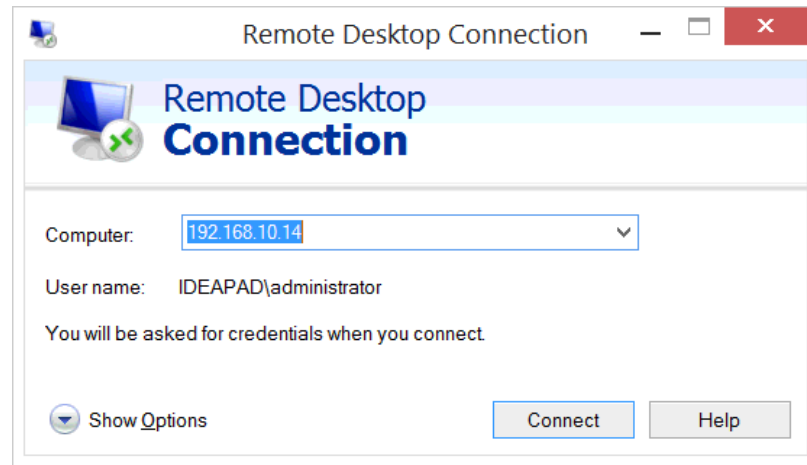


Figure J.2. Remote desktop client screen

3.3. Log in the Windows 2016 system with the following credentials:

- User: Administrator
- Password: 2hLMmVGt

3.4. After log in the Windows 2016, complete the next actions inside this system:

- 3.4.1. Open a command prompt window by double-clicking on the icon “Command Prompt”, located at the Windows 2016 Desktop.
- 3.4.2. Select two commands from the next list and execute them, inside the command prompt window, once at a time:

cls	ipconfig	route print	dir	cd
date /t	hostname	netstat	tree	ver
time /t	vol	whoami	arp -a	dispdiag

3.4.3. Minimize the command prompt window. Do not close it. If it was closed by mistake start again from step 3.4.1.

3.5. Once the previous actions are completed, leave the remote desktop connection open (do not close the session, just minimize it to continue with the next step).

4. Connect to the MySQL server running on the VM:

- 4.1. Open the application MySQL Workbench by double-clicking on the icon “MySQL Workbench”, located at the Desktop.
- 4.2. Go to menu “Database” and then select “Connect to Database...”.
- 4.3. Select the stored connection “VM4: Windows 2016” that was previously created to connect to the IP address 192.168.10.14, port TCP 3306, and using the root account. Then click on the button “OK”. Figure J.3 displays the MySQL Workbench screen and the aforementioned stored connection.

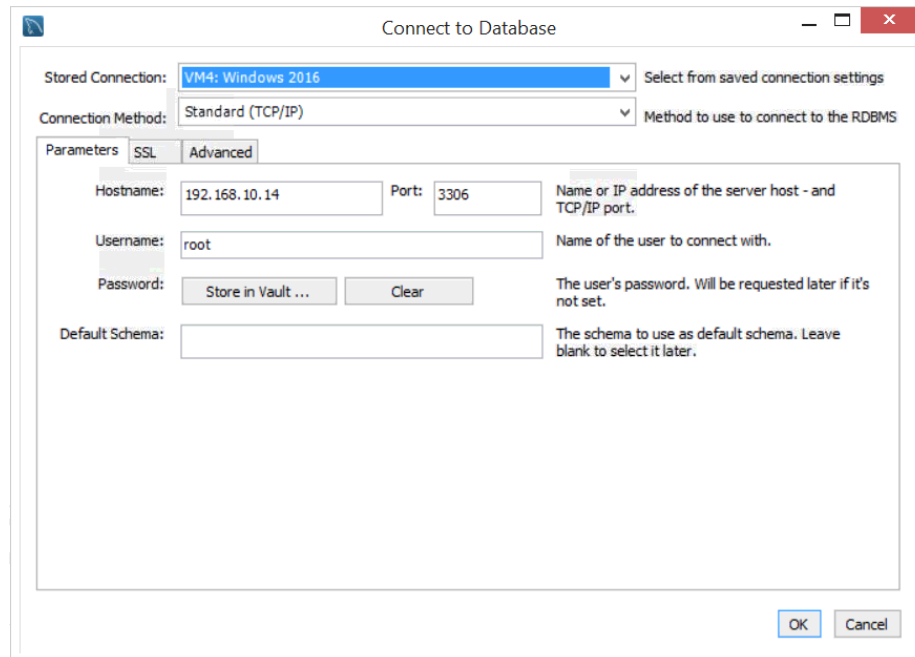


Figure J.3. MySQL Workbench screen

- 4.4. Wait until the MySQL connection is established. Figure J.4 illustrates the MySQL Workbench screen at this point, after the MySQL connection is established and ready to accept SQL statements.

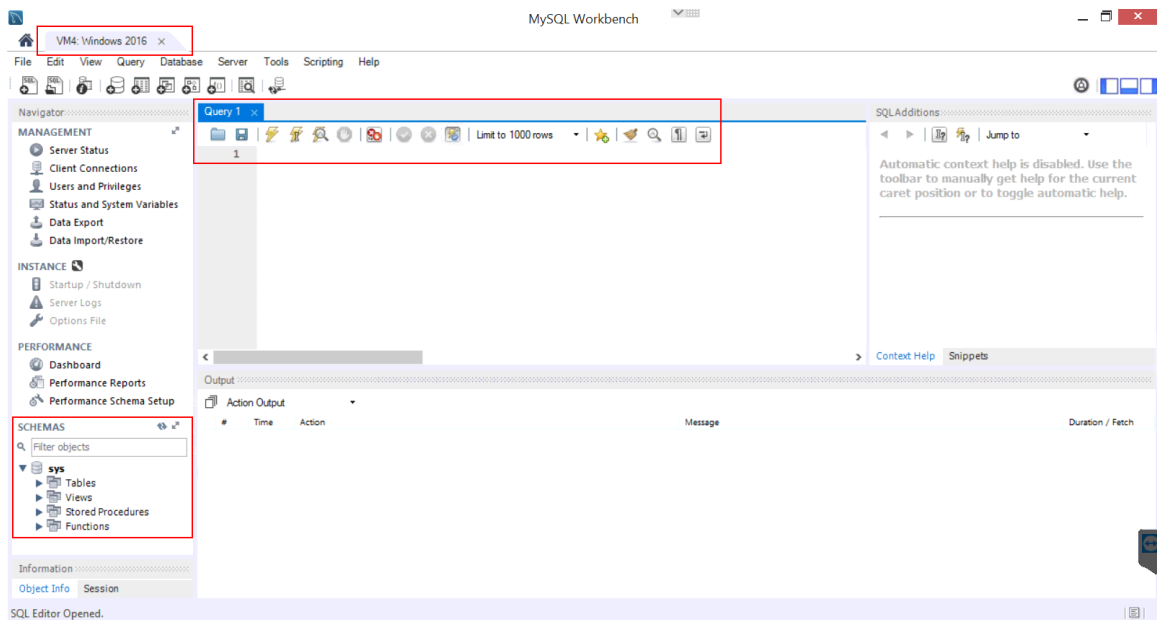


Figure J.4. MySQL Workbench screen once the MySQL connection is established

4.5. Close the application MySQL Workbench.

5. Complete the first available row in the next table, detailing the date and time the interaction was performed, the X value selected at step 2.2 and the commands executed at step 3.4.2.

Test	Date	Time	X value	Commands executed
1	05/18	6:10pm	9	tree – arp -a
2	05/20	7:30pm	8	whoami – tree
3	05/21	6:50pm	8	route print – dispdiag
4	05/22	6:35pm	9	ver – netstat
5	05/23	6:30pm	8	hostname – ipconfig
6	05/25	4:30pm	7	netstat – tree
7	05/30	6:20pm	2	dir – whoami
8	06/01	7:30pm	3	date /t – ver
9	06/05	11:30am	5	cls – dispdiag
10	06/08	10:55am	9	ipconfig – hostname