UiO : Department of Informatics
University of Oslo

Agentless Inspection of Virtual Hosts Configuration by Interaction Through The Virtual Hardware

Andreas Liaker

Master's Thesis Autumn 2014

Agentless Inspection of Virtual Hosts Configuration by Interaction Through The Virtual Hardware

Andreas Liaker

12th December 2014

Abstract

In this thesis the possibility to interact with the virtual machine without traversing the network or using monitoring agents are explored. Instead of using this traditional approach the intention are to exploit the possibilities which are present when a System Administrator does control the hardware that host the virtual machine. In the research community this has been discussed under the topic of introspection, and there has been considerable research in this area.

The contribution don by this thesis is the creation of tool that can analyze the memory of a targeted virtual machine from the virtual machine monitor. The focus of this analyzes is to determine if the targeted virtual machine is compliant according to a predefined configuration stored in more secure location.

In addition to the development of the tool, the performance impact on the targeted virtual machine is investigated. One of the intentions of the development of the tool in this thesis is to remove the load caused by a traditional agent. Instead these workloads are transferred to server that has privileged access to the targeted virtual machines hardware.

In the end the results are discussed and it is concluded that a prototype is developed and the performance impact on the targeted virtual machine are acceptable. Because there is only a prototype that is developed improvements are suggested in additions to future work.

Contents

List of Figures

List of Tables

Acknowledgements

I would like to express my gratitude to the following people:

- My supervisor Andrew Seely for his valuable advice, guidance, enthusiasm for my thesis, keep me motivated when things went wrong and reviewing my final work.
- My Company Statsbygg and my Manager Hanne Flostrand for supporting my education with necessary equipment and time to study.
- My coworker Kenneth Gudem for valuable advice with problem-solving and interesting discussions.
- My Wife and two kids for patiently supporting my work and still loving me.

Chapter 1

Introduction

The internet is no longer a place where friendly scientists are sharing ideas and information. Today this friendly society is turned into a place where nations are waging secrets wars, and international terrorist and criminal organizations are coordinating their illegal efforts. The enemy is hidden and the motivation of his actions is unknown. The victims can in many cases seem to be arbitrary, so any organization or person could be the next victim.

As the attacker has changed so has also the actual attack. In the beginning when the attacker was a kid still living at his parents place, the attack was to penetrate the security of a system and take it down or change a webpage to something completely else. This kind of attack was easier to defend against, or at least you knew that you ware compromised. Now that the attacker has evolved into a well-financed organization the attack has become more sophisticated as well. Today an attack can penetrate a system and lay dormant until the right opportunity arises.

This new attack often has the capability to hide from security solutions like intrusion detection systems [13]. The first step of an attack is in many cases a port scan, but instead of rapidly going through all the ports, attackers have learned to camouflage an attack by blending into legitimate traffic like using more time between trying each port. This approach is difficult for a network intrusion detection system (NIDS) to detect, which has resulted in the creation of host intrusion detection systems (HIDS). But the HIDS is only capable of defending against attacks after a system is affected, so the attacker mask their presence by installing root kits which hide the malicious processes.

When a person investigates the threats that are out there, and how fast the malicious attacks are adapting to new security solutions, it is easy to draw the conclusion that resistance to attacks is futile. For instance for a security company to develop a signature against a virus, some systems first need to be infected. And when a new vulnerability is discovered there is first a need to develop a patch and then the patch need to be tested before it can be implemented. The length of time from when the vulnerability [12]

is discovered to a successful patch can be significant. In addition there is a valid point that the requirements for testing malicious code that will exploit a vulnerability is not always as important as testing the actual patch that will seal that vulnerability.

Counteracting these threats requires a significant amount of resources. And there are often conflicting interests when the subject about using resources on security is discussed. One of the reasons for this is that it can be hard for non-technical decision takers to see the benefits of investing in security when it is weighted against new functionality that has promises of increased profit. Due to this a system administrator has a difficult task when it comes to decide on security solutions.

A technology that has released a considerable amount of resources for the system administrator in the past is virtualization [35]. Virtualization has simplified the management of the server infrastructure, and enabled the system administrator to utilize more of the potential of the already purchased hardware. One of the objectives of this thesis is to enlighten and prove that there is more potential in this technology than is commonly in use today.

A topic in the virtualization paradigm that has received a considerable amount of scientific attention is the concept of virtual machine introspection (VMI). VMI is to directly analyze the state of virtual machine hardware like memory and disk, from a secure location, without using an agent or rely on the guest operation system API. This secure location is in most cases the virtual machine monitor (VMM), or a virtual machine that has privileged access to the guest virtual machines hardware like the Dom0 server in a Xen architecture. The concept of VMI is described in detail in section Virtual Machine Introspection 2.5 on page 13.

1.1 Motivation

A legitimate question when it comes to VMI is why is it interesting to use this technique when it is clearly much more complicated than the traditional way with using an agent or directly run system calls to the Operation System Application Programming Interfaces (OSAPIs)? The answer to this is yes there is more complicated but it also enables new capabilities that are not possible with the standard techniques. And there are reasons to believe that some of these capabilities are not yet discovered.

The first benefit of using virtual machine introspection is that it is difficult for the guest operation system (GOS) to be aware that it is actually being monitored. There are some indications that might be taking place, but this is mostly based on the assumption that if the OS is running on a virtual machine it might also be monitored with VMI techniques. For the

operation system to detect if it is actually a virtual machine or a physical one, it is often enough to investigate the hardware. Like virtual hardware has in many cases specific virtual hardware drivers, or virtual CPU will actually reveal it is virtual if asked. But even if the GOS knows it is a virtual machine, it can only assume it is being monitored, and it is difficult to know in which way.

The VMI capability of hiding from the GOS is also valid for hiding from malicious code that has infected the GOS on the virtual machine. This capability has resulted in research on numbers of VMI security solutions which is explained more thoroughly in Related Work section 2.7 on page 13. The idea is that it is difficult for the malicious code to hide from the VMI monitoring tool when the VMI tool does not depend on any code in the environment the malicious code controls. For instance it is common for malicious code to install a stealth root kit on the machine it controls, to hide the malicious processes. So when you execute a code that will list all processes like this command in an Ubuntu OS.

```
# sudo ps aux
```

The code usually list all processes, but with a stealth root kit it lists all processes except the processes that the malicious code wants to hide. This might be a listening process which intends to keep a backdoor open for later use. The VMI monitoring tool do not depend on executing the command that are compromised, instead it analyze the memory where the malicious code need to run in order to work.

This is not only valid for processes, but also for anything that needs to be in memory, like the running configuration of an application. A simple example is if you run this command in an Ubuntu OS.

```
# sudo ifconfig
```

This will list your active network configuration. There is a simple task for a malicious code to tamper with output to show something different than what is actually the running configuration. But even if the output is altered, the running configuration needs to be in memory. And this is true for all processes running on an OS, and a VMI monitoring tool should be able to detect this.

In many IDS and VMIIDS solutions the detection methods are based on recognizing patterns and malicious behavior. In order for the IDS or VMIIDS solution to detect this patterns or malicious behavior it is necessary to have some knowledge about this up front, and this is not necessary a trivial task to accomplish. A thing that is more trivial is to gather knowledge about the configuration of a system. So a VMI configuration compliance system which detected drifting configurations cud be a god

supplement to a VMIIDS solution.

A known malicious code that has used the ability to report the wrong state to the user is the computer worm stuxnet [27]. Simply explained, this computer worm is believed to be designed for attacking the Iranian nuclear program, by destroying the centrifuges used for separating nuclear materials. The computer worm destroyed the centrifuges by letting the centrifuge spin outside their thresholds and tear them self apart. This was done without the control software reporting any failures. With a VMI tool this should in theory be possible to detect that the control software did not use the correct parameters.

In addition to the security benefits of a VMI monitoring tool, there are other factors that can motivate the development of this kinds of tool. Some of these factors are in-guest performance, and the way it is deployed. By deploying the VMI monitoring tool somewhere outside the virtual guest, most of the code execution will happen outside the virtual machine, and free up in guest resources. This will in most cases be executed on the same physical hardware, but the resources dedicated to a virtual machine can be used by the guest operation system and it applications instead of security tool agent.

When a tool is abstracted from the virtual guest operation system and deployed on a machine that has privileged access to the physical hardware like the VMM or Dom0 in Xen, the design is simpler. A VMM can host multiple virtual machines on one physical machine, so the VMI tool will only need to be deployed one time for each physical machine. The VMI tool can then be configured to interact with all the virtual machines that are running on the given VMM. This simple design will also simplify the process of updating the tool, like new definition files.

1.2 Scope and Problem statement

In this thesis I will develop a prototype for a tool that can be used to identify if a virtual machine is compliant according to a pre-defined set of configurations. The prototype will use VMI to analyze the memory of the targeted virtual machine, and then compare it with the desired configuration stored on a secure place accessible by the prototype. There is only the running configuration or the memory that will be analyzed and not the actual configuration file. To analyze the actual configuration files might be interesting, but is left out of scope because there is only the running configuration that are considered criteria to determine if a virtual machine is compliant or not.

If a virtual machine complies with the desired configuration, the virtual machine will be reported to be compliant. But if the virtual machine does not comply with one of the parameters described in the desired con-

figuration, it should be reported as an uncompliant machine. Based on this result the virtual machine can be connected to the network if it is compliant and disconnected if it is not compliant. The prototype can be executed on a schedule to do compliant test regularly in case the configuration on the virtual host has changed and then correct actions can be performed.

The idea of this behavior is to not only to protect against malicious code. In addition this can be used to protect against misconfiguration as well. There might be reasons for other behaviors in case for misconfigurations, but this is considered out of scope for this thesis.

In addition to this functionality the prototype should not impact the guest virtual machine performance. As described in the Motivation section 1.1 on page 2, one of the reasons for using VMI tools is to free up guest virtual machine resources.

To summarize the capabilities of the prototype:

- Determine if a virtual machine is compliant with the desired configuration.
- Connect or disconnect the virtual machine according to it compliance status.
- The ability to re-run the test in case of a compliance change.
- Negligible performance impact on the guest virtual machine.

1.2.1 Problem statement

Using Virtual Machine Introspection (VMI) to validate if a virtual machine is complaint with respect to confirmation management and malicious software standards before being allowed to access a production environment.

1.3 Challenges

The first of the challenges that will be addressed in this thesis is how to interpret the memory from the virtual guest. When you are operating from inside of a virtual machine, the operation system has a set of commands and APIs that is available to help identify where in the memory the data is located. When you are operating from the VMM or a server that has direct access to another machines memory the memory is just a stream of continuous data. This behavior is named the Semantic Gap [36][8], and is explained in more detail in the Semantic Gap section 2.6 on page 13. To bridge this gap is not only a challenge for this thesis but also for most other VMI research.

Another challenge that needs to be address is the performance impact the prototype will have on the virtual machine. In this thesis the Xen server will be used, and the VMI tool will be implemented in the dom0 server as described in Planning the Project chapter 3 on page 17. This will have an additional performance impact, due to the fact that the dom0 server needs to contact the VMM to get access to the virtual machine memory. So for this thesis it will be important to design the prototype with the goal of having a small as possible performance impact of the monitored virtual machine.

In addition to these challenges it is important to consider the state of a virtual machine during a boot. When a virtual machine is powered off it has no configuration located in the memory. So it needs to be decided if a machine should be allowed access to the production environment initially, or if it should wait until the first analysis of the virtual machine is performed. The last option will of course result in a longer time before the virtual machine is available during a boot, depending on the frequency the prototype performs a compliance check on the virtual machine.

Chapter 2

Background and Literature

2.1 Server Virtualization

The idea of sever virtualization is to abstract the operating system from the physical hardware, and instead put in place a Virtual Machine Monitor (VMM) which will control the hardware and make it available for one or more guest operating systems (GOS). Because most systems only use a portion of the resources available [30], this architecture of serving more than one operation system makes for a more efficient use of the available hardware resources. Generally there is to types of virtual environment [26][18][1]:

- Type 1. The VMM runs directly on the hardware
- Type 2. The VMM runs on top of another operating system

2.1.1 VMM type 1

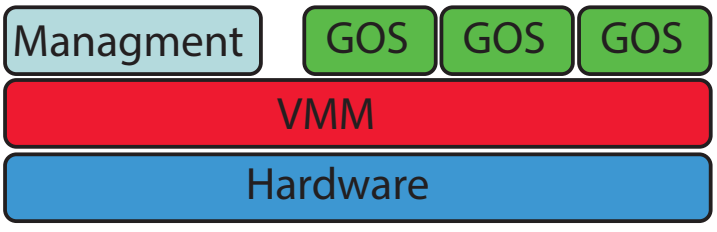


Figure 2.1: Type 1 VMM

For type 1 VMM, a layer of abstraction is removed and the VMM runs directly on the hardware as illustrated in figure 2.1 on page 7. When a VMM runs in a type 1 environment it is often referred to as a hypervisor [1], and is the architecture used by VMware ESX, XEN Server and Microsoft Hyper-V. This is commonly used for enterprises production environments,

and with the use of paravirtualized or native drivers this also gives the best performance. See Paravirtualization section 2.3.2 on page 10 and Hardware assisted virtualization section 2.3.3 on page 11.

2.1.2 VMM type 2

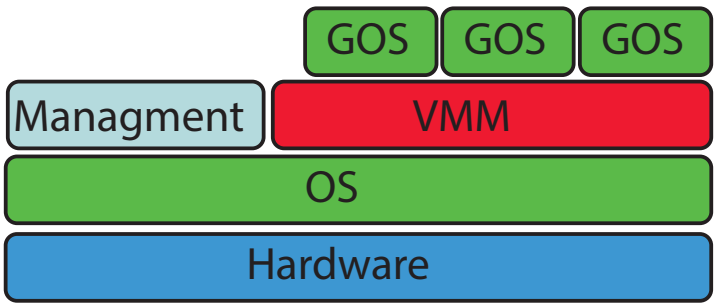


Figure 2.2: Type 2 VMM

In a type 2 environment the VMM does not run directly on the hardware, but runs on top of an operating system. See 2.2. This type of environment does generally not perform as well as a type 1 because the VMM needs to use the underlying operating system as an interface to the hardware. Sometimes a type 2 VMM is referred to as desktop virtualization, and some examples of a type 2 VMM are VMWare workstation [31] and Virtual Box [21].

2.2 Protection ring

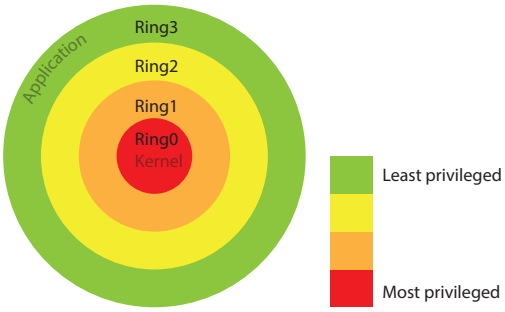


Figure 2.3: Protection Ring

To explain how virtual machines get access to the hardware it is helpful to first explain the concept of protection rings. The protection rings are in the x86 architecture a mechanism enforced by the CPU to protect the system from failure [4]. The x86 architecture consists of four protection rings usually numbered from zero to three, where ring zero has the most privileges and then the privileges decrease until the third ring which has the

least privileges. See 2.3. For an ordinary operating system ring 1 and 2 is not used, while the kernel runs in ring 0 and the applications runs in ring 3.

The concept of protection rings is to protect against arbitrary usages of inner rings resources, and instead provide a predefined gateways for accessing these resources. This will prevent an application from a less privileged ring to mis-use resources from a more privileged ring.

Because type1 VMM and guest operation system kernel both expect to run in ring0, recent CPUs from Intel and AMD have implemented hardware virtualization assisting capabilities. Intel implemented Intel VT-x and AMD implemented AMD-V. Both of this technologies implement a Ring -1 layer which enables a VMM to control ring0 access, which then enables operation system to run ring0 natively without interfering with other virtual hosts.

2.3 CPU virtualization

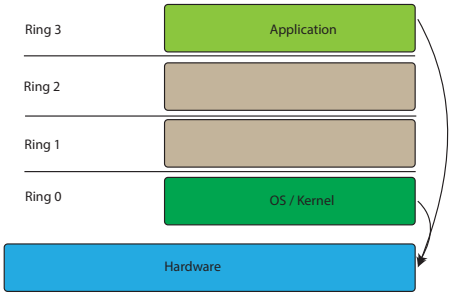


Figure 2.4: Without virtualization

As mentioned in 2.2, an operation system is designed to run directly on the hardware and need to run it most privileged instructions in ring0. The instructions that do not need to be executed in ring0 are usually carried out in ring3 such as user applications. This is illustrated in figure 2.4. With the operating systems need for owning the ring0 becomes a challenge for virtualization when this depends on putting a VMM underneath the guest operation system. To overcome this challenge the different virtualization vendors have embraced different approaches, including: [22][35].

- Full Virtualization
- Paravirtualization
- Hardware assisted virtualization

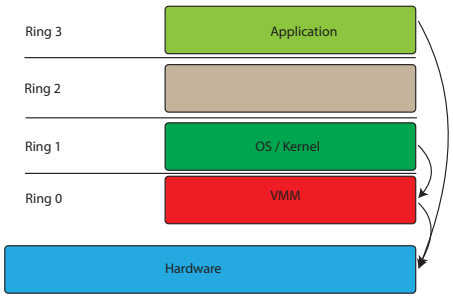


Figure 2.5: Full virtualization

2.3.1 Full virtualization

In a fully virtual environment the guest OS kernel is not aware of it running in a virtualized environment, and all the guest systems hardware is virtualized. As illustrated in figure 2.5 the VMM is running in ring0 and the guest operation systems privileged kernel instructions are executed in ring 1. Because these instructions expect to be executed in ring0 the nonvirtualizeble instructions are binary translated to new sets of instructions that have the intended effects on the gust operation systems virtualized hardware. Fore higher performance the user mode application instructions are executed directly on the processor. This type of virtualization does not require any modification of the guest operation system kernel, and may trade some performance for maintainability.

2.3.2 Paravirtualization

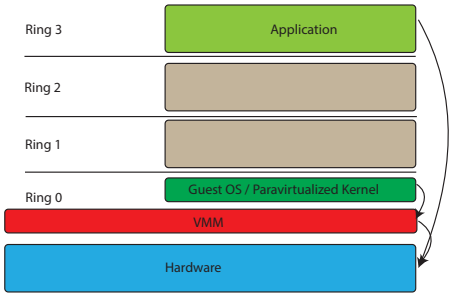


Figure 2.6: Paravirtualization

With paravirtualization the kernel is made aware that it is running on a virtualized environment, which allows it to run in ring0. Because most operating systems are not designed to run in a virtualized environment this requires some deep modifications of the OS kernel. This modification enables the guest operating system to do system calls that are difficult or slow to virtualize, directly to the VMM/hypervisor (hypercalls) [6]. The paravirtualization architecture of doing the privileged system calls with hypercalls

is faster than the fully virtualized architecture [35]of doing this call with bit translations or fully software emulation.

While development of bit translation is considered to be very complicated, to modify the guest operation system kernel and develop the VMM to accept hypercalls is a simpler task. The drawback of this approach is that the Guest operation system needs to be modified, and this has its challenges when it comes to proprietary operating systems like Microsoft Windows.

2.3.3 Hardware assisted virtualization

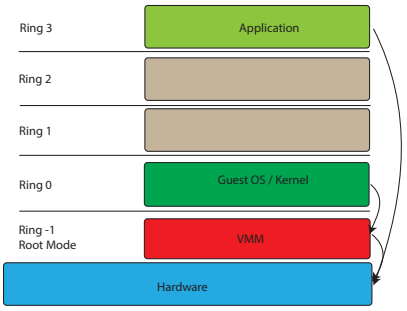


Figure 2.7: Hardware assisted virtualization

In 2006 Intel released VT-X and AMD released AMD-V which enables the concept of hardware assisted virtualization for the x86 architecture [9]. When one of these features is enabled a new CPU execution mode beneath ring0 is available. See figure 2.7. This level is often named ring-1 or root mode, and when this mode is enabled all the privileged and sensitive calls are sent directly to the VMM. When the VMM runs in root mode the need for paravirtualized OS kernels or binary translation is eliminated.

Hardware assisted virtualization has a range of different names. It is known as accelerated virtualization, Hardware virtual machine (HVM), and native virtualization. HVM is the name that is used by the Xen technology that will be used in this thesis. The benefit of using hardware assisted virtualization is that the need for modifying the kernel and bit translation is reduced. The first attempts to use hardware assisted virtualization gave little performance advantages over software emulated hardware, and in some cases the performance was worse [9] [22]. To improve this a hybrid solution is used where optimized paravirtualized drivers boost performance while hardware assisted virtualization eliminates the need for modifying the kernel. In Xen this is called PVHVM or PV-on-HVM [35].

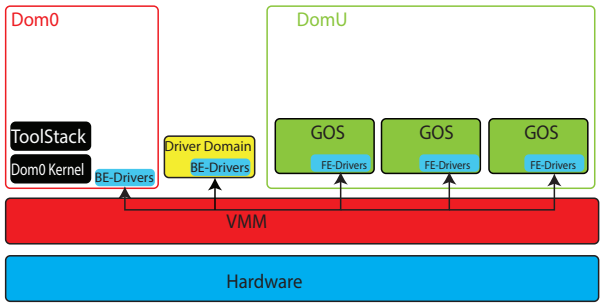


Figure 2.8: Xen Server Architecture

2.4 Xen Server Architecture

Xen server is a type 1 open source VMM, and it supports both paravirtualization PV and hardware assisted virtualization which they have named hardware virtual machines (HVM). The VMM or hypervisor are one of the smallest type 1 VMM that is available today, and has a size of about 1 MB [35]. One of the reasons for this small size is that it does not contain any device drivers, instead the drivers are present in a special virtual machine known as Domain 0 or a driver domain server [5].

Xen server has servers running in two types of domains, which each has a separate set of privileges. First there is the control domain or Dom0, which is a special virtual machine that has the highest privileges. The Dom0 server has direct access to the hardware and handles all the interactions with the systems I/0. It also handles all the communication between the guest virtual machines, and the outside world. This architecture results in the Dom0 server being essential, and the VMM will not work without this server.

The control domain requires a server that has a Xen-enabled kernel, and most Linux distributions that are based on the recent Linux kernels have this support. Because the Dome0 server that controls the drivers is based on a Linux system the support for hardware is quite large. As indicated earlier the driver control can be delegated to a driver domain server, which will only have the privilege to control the hardware it has been delegated. This design will free capacity on the Dom0 server to lower the risk of it being a bottleneck. In addition this will enhance reliability because the driver that may be more prone to failure then the rest of the OS is isolated in a separate unprivileged server.

In addition to the control Domain the Xen architecture consists of a domu domain which contains all the guest virtual machines. All the guest virtual machines has a set of front-end drivers witch are given access to the hardware through the back end drivers in dom0 or driver domain. See figure 2.8. To deploy and control the virtual machines in domain U the dom0 server uses the tool Stack.

2.5 Virtual Machine Introspection

Virtual Machine Introspection (VMI) is used in most cases to passively or actively monitor an exposed unsecure guest virtual machine (GVM) from a secure isolated machine, by tapping into the GVMs virtual hardware like memory, CPU register and disk. When VMI is performed there is no need for the typical agents, because the monitoring occurs directly to the virtual hardware from the entity that controls this hardware. In the case of a Xen server it will typically be the VMM or the Dom0 server that will monitor the guest virtual machines.

2.6 Semantic Gap

The Semantic Gap [36][8] is the difference between how the Operation System on a virtual machine can access the systems resources and how it is represented for the VMM. Inside a virtual machine the Operation System can access the systems resources through the virtual interface in the same way it access the physical hardware if the machine that is not virtualized. From the VMM the same resources are presented as a raw stream of data, and the VMM has little understanding of the semantics of this data.

It is to bridge this Semantic Gap problem much of the VMI research tries to solve. The VMI applications develop is either semantically aware or unaware [18]. A semantically aware VMI application has an initial knowledge about the Operation System [13][10], like LibVMI [14] which uses the system.map file see Configure Libvmi Appendix A.5 page 55. If the VMI application is semantically unaware it builds the knowledge about the virtual host over time[8].

2.7 Related Work

The concept of virtual machine introspection was first investigated by T. Garfinkel [23]. He presented the idea of moving an IDS solution out of the guest operation system and down on the virtual machine monitoring (VMM) level. By this approach he manage to get much of the host HIDS visibility without the vulnerability of running the IDS on the same host that it is inspecting.

It is possible to divide the research that has been done about VMI into two categories. One of the two categories is pure monitoring or read only. The second one is the interactive one, that not only monitors the guest VM but interact with it as well. The first category is the most common one, but

the recent years I have seen more research that belongs to the second category as well.

One example that falls under the category of pure monitoring is Antfarm [10], which is technique that can be used to track processes inside a virtual machine. The process inspection have been taken a step further in lycosid [11] where they are trying to find hidden processes by using hypotheses testing. Many samples are taken over time, in a busy system, and the probability for hidden processes is calculated.

Another example of monitoring-only solutions is VMI-Honymon [13] which is an intrusion detection system that uses memory based introspection for monitoring honeypots. The monitoring with VMI is also transferable into the cloud, and this is explored in NFM [28], which is a solution for monitoring in the cloud. The idea is to enable subscription on monitored data even if the actual system is down.

In the category of more interactive VMI techniques, the level of interaction is varied. Like in Manitou [16] the solution inspects the code that is going to be executed inside a guest virtual machine. Then Manitou runs on the VMM and uses the per-page-permission bit to determine if the code that is going to be executed corresponds to the authorized code. To authorize the code Manitou uses a cryptographic hash of the code in the moment before it is going to be executed.

Another approach is explored in IntroVirt [12]. This tool uses predicts to discover vulnerabilities and prevent them. A predict is a type of virtual patch that is developed by a person with the same skill set as the person that develop the actual patch. This predict can then be used to protect a system until the system is tested and updated with the new patch.

A solution that have taken the level of interaction on step further is Exterior [7]. Exterior is a prototype that can execute a command from a secure virtual machine (SVM) and through VMI insert the code to a guest virtual machine (GVM). The code can then be executed on the GVM without any privileges on the GVM. This demonstrates the possibilities that are possible with VMI as well as the security concerns.

When the interaction with the machine is moved from the operating system inside the virtual machine to the VMM layer, the challenge about the “Semantic Gap“ rises. When inside the virtual machine there are system calls and API‘s [15], while at the VMM layer there is only a bit structure to interact with. This Semantic Gap is a great challenge, and in order to overcome this, a great understanding of the operating system is needed [3]. And with the wide variety of operating systems that exist today, there is a good deal of operating systems to have a great understanding about.

Fortunately there has been some work done to decrease the Semantic

Gap. The common denominator for many of the approaches is that in some way the behavior of the operating system is inspected and recorded when it is exposed for specific event. This event may be for instance an execution of a command. In Virtuoso [3] they automatically create introspection tools, by analyzing traces of small in guest programs/commands that compute the desired introspection information. This small program can then be executed on a secure virtual host to retrieve the given information. Exterior has another approach [7] to accomplish the introspection. Instead of recording the behavior of the operation system up front, exterior performs the analyzing of the OS behavior real-time from a secure VM. The drawback of this approach is that an identical machine to the guest VM is needed to serve as the secure VM for each different guest operating system.

Another approach that is suggested is [2] to combine forensics memory analyzing (FMA) with VMI to overcome the challenges of the Semantic Gap. The suggested technique is to make a WMI tool access the live memory of a guest VM as memory dump file. This " live memory dump file" can then be used by the FAM tools to do its analyzing. This approach limits the possibilities, but the gain is that many of the already complete FMA tools can be, with some modification, used in a VMI solution.

The main motivation for much of the research that has been done in the field of VMI is the different security possibilities it enables. One of the features that attract attention is the possibility to monitor virtual host, with almost no footprint. It is suggested [18] that it might be possible for an attacker to discover that the host is being monitored by processes using more time than expected, but this is a difficult task. An IDS solution was suggested with the prototype Livewire [23], which explored the benefits of taking an HISD out of the VM, and instead monitor with VMI. The IDS solution honeymoon [13] uses this stealthy VMI monitoring capability to deeply honeypots which they monitor in the intention to learn the behavior and tactics of the attacker.

One of the indications a VDI-IDS solution is searching for is the presence of a stealth root kit. One method to detect rootkits is to investigate what is called a cross view. On view is the one you get from the operating system, or the untrusted view. The other is the one is obtained from VMI techniques which in unlikely that is tempered with by the attacker. These techniques are used by both Livewire [23] and honeymoon [13], but they do not take into an account that this is a small time delay between the capture of the two views. In this small time difference a process may spawn or vanish. As mentioned earlier in lycosid [11] they trade accuracy for time and calculate probability with hypotheses testing.

In addition to IDS solutions there is also other security solution that is interesting in a VMI perspective. One of these solutions is the one explored in introvert [12] where they create predictions which can be used to prevent a system from known vulnerabilities until the system has received the

necessary security updates. There is also one commercial counterpart to introvert named Deep Security from Trend Micro[29]. This solution has a number of functionalities, including IDS/IPS, web application protection, application control and firewall protection. Deep security is designed to run on VMware, and depends on vshield [19][32] which is a tool from VMware that can be used for virtual machine inspection.

Chapter 3

Planning the Project

The goal of this project is to use virtual machine introspection to validate if a virtual machine is complaint to a given policy. To have proof of concept I will first need a virtual environment. To keep the environment as simple as possible and still be able to prove the concept it will be sufficient to deploy only one physical host. In a true production environment there is in most cases more than one physical host, but then it should be sufficient do duplicate the prototype on each physical host. In the case of multiple physical host architecture there will most likely be a need of a central administration point, but this is out of scope of this thesis. With minor modification to the prototype presented here, it should be possible to make this a part of an existing monitoring tool like Nagios.

For the selection of a virtualization technology I will use Xen Server 4.4.0, with an Ubuntu 14.04 Ubuntu server as dom0. There are multiple reasons for using Xen server as the virtualization technology. For example there is a widely used open source type 1 VMM, and there has been considerable research on VMI on Xen Server. To select a type 1 VMM are preferred because there is the most common solution for production systems, and in a production environment the need exists for VMI compliance tool. In addition to the research that has been don on VMI for Xen, the Semantic Gap bridging tools are available on this platform.

The tool that will be used in this thesis to bridge the semantic gap is libwmi [14]. This tool is mainly developed to analyze the memory of the virtual machines. This will apply to the solution very well because when the configuration is analyzed in memory it will in most cases bee the active configuration that is analyzed. It is possible for malicious code to camouflage the running configuration, but the running configuration needs to be in memory.

The libvmi tool is designed to be implemented in the dom0 server, and not in the VMM which is another option for a VMI tool. The drawback of the dom0 placement is that there is some performance delay due to the VMI tool needs to contact the VMM to get access to the memory, and cannot do

it directly. But for this architecture there is no need to modify the VMM and the code of the VMM can be simpler and smaller. When the code is kept small and simple there is less chance for bugs and vulnerabilities, and by this a more secure VMM.

To directly use libvmi to analyze the guest virtual machines memory requires significant knowledge about how the operation system and how it takes advantage of its memory. This is considered difficult knowledge to acquire, and will limit the possible candidates to further develop the prototype. Ideally it would be desirable that a system administrator could be able to customize and adapt the prototype to cover his needs.

A solution to this challenge is to use Volatility [33] with the PyVMI [24] plugin which enables Volatility to interact with virtual guests through the LibVMI library as suggested in [25]. Volatility is a well-developed open source memory forensics tool, whit easy to use and an active development community. The drawback of using this is that Volatility is design to analyze a memory dump, and is not able to take advantage of all the capabilities that libvmi provides. This drawback is considered secondary because the main purpose of this thesis is to prove that there is possible to investigate if a virtual guest is compliant with the use of VMI techniques.

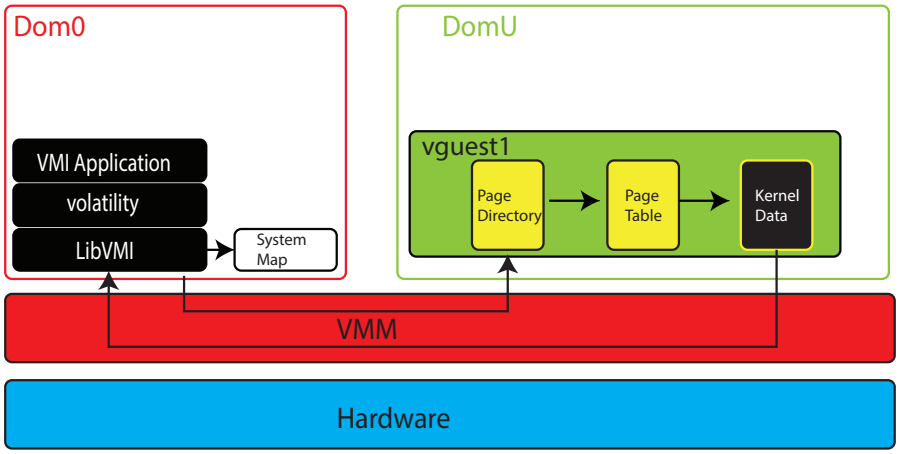


Figure 3.1: The project environment

In figure 3.1 the designed is illustrated. LibVMI is installed on the dome0 server. In addition the system.map file from the guest virtual machine in DomU need to be copied to the Dom0 server. This file will be used to find the virtual address of the kernel symbol [17] in the guest virtual machine. Libvmi will inspect the kernel page directory to find the page table which contains the requested data, so it can be returned from LibVMI to Volatility which then can supply the prototype with the running configuration of the inspected virtual host in DomU.

3.1 The prototype

The prototype that will be developed in this thesis will be written in the Perl programming language. Perl was selected because it is widely known among system administrators. Perl has also good features to manipulate the feedback form the volatility application, and will probably solve the needed task in satisfying matter. The functions that the prototype will need to solve are the following.

- Read the desired configuration from a configuration file.
- Test if the virtual guest is compliant according to the desired configuration.
- Give a feedback if a virtual host is compliant or not.

In the problem statement I stated that the prototype will detect if a virtual host is compliant or not to be allowed access to the network. The prototype will only report on the status, and will not do the actual attach to and detach from the network. The reason for this is that during the experiments that will be conducted in this thesis it is necessary to have network access, and the compliance status will be interesting as well. There should be quite easy to adapt this behavior by calling on this to commands in the xl tool stack.

```
# xl network-atach
# xl network-detach
```

The primary objective of developing this prototype is to prove that it is possible to test if a guest machine is compliant, by using VMI techniques. In addition it is important to prove that there is possible to test the running configuration in the memory. The parameters I have selected to test is listed below.

- Test if a GVM has the correct IP configuration.
- Test if a specific process is running on the actual GVM.
- Test if the GVM has the desired kernel version.

3.2 Test Plan

3.2.1 Experiment 1: Accuracy

When the prototype is developed there will be necessary to determine the accuracy of the application. I will go through the different parameters the prototype is intending to test, and provoke a failure and then validate that

the prototype acts accordingly. One of qualities of the prototype is that it only will test against the running configuration, and not any configuration files. To validate this behavior I will change the in memory configuration and the configuration file. Then I will validate that the prototype only reacts when the in memory configuration is changed. In the table 3.1 I have added the expected results when a parameter is changed to an undesired configuration.

	In memory	File
IP address	Not Compliant	Compliant
Running process	Not Compliant	NA
Kernel version	NA	NA

Table 3.1: Expected result for compliant test.

To test the IP address I will first edit the network configuration file and make sure the prototype still reports the virtual machine to be compliant. Then I will remove the changes in the network configuration file, before I use the ifconfig command to force another IP configuration on the virtual guest. Then the prototype is expected to report an uncompliant virtual host even though the network configuration file is correct. This can be related to any application which gets it configuration from a configuration file. By doing this simulation I simulate a malicious code altering the running configuration even it cannot be detected by a monitoring tool that reads the configuration file. This will also simulate a stressed system administrator changing a configuration without following the correct change procedures.

For a running process I will only test to stop the running process to determine if the prototype report this as an uncompliant virtual machine. To provoke a change to the kernel it would be possible to upgrade the virtual host to a newer version. The problem whith this is that the System.map file will be changed and then LibVMI and Volatility will not be able to read the memory and analyze it. The prototype will still report the virtual host as not compliant, but this will be because all the tests are failing, and not because it detected an undesired kernel version.

3.2.2 Experiment 2: Performance impact on virtual guest.

In addition to testing the accuracy, it is also important to measure the performance impact inside the guest virtual machine. In most production environments it may be enough to run the compliance test once every five minutes or so. A test running in this pace should ideally have almost no performance impact on the guest virtual machine, but it would be interesting to find an estimate of what the actual performance impact would be.

It is possible that in some scenarios the one compliance test for every five minutes is not enough, and it might be necessary to have a more real

time compliance test. It is difficult to know up front how fast it is possible to do the compliance test, but during the experiments I should try to get close to this threshold. In addition to getting close to the threshold of possible test, it will be interesting to see if there is any breaking point on the guest virtual machine. The breaking point in performance will be if there is a rapid increase in the performance impact on the guest virtual machine for a specific amount of compliance testing per minute.

In addition, I will not tax the Dom0 to much so I will need to measure the execution time on the dom0 machine as well.

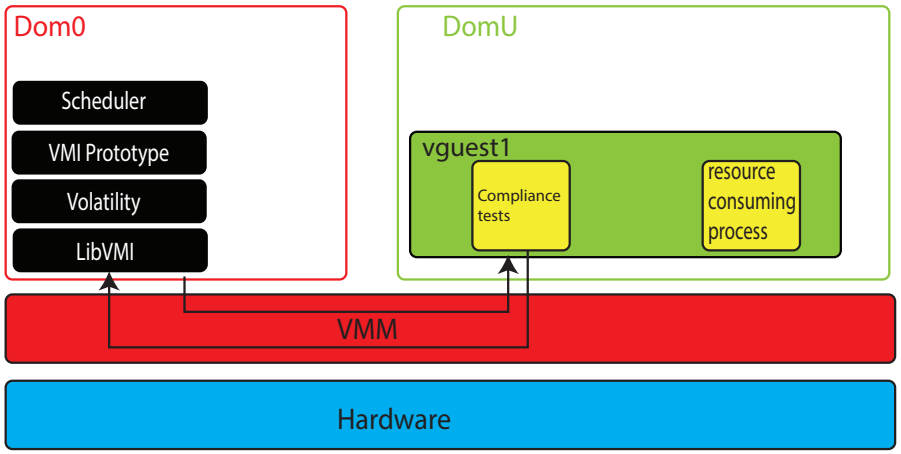


Figure 3.2: Experiment 2: Performance impact on virtual guest.

To perform the experiment I will create a resource consuming process on the guest virtual machine and then measure the time it do take to complete the test. First I will create a baseline by running the resource consuming process with no compliance test being done by the prototype. Then I will re-run the resource consuming process but simultaneously I will run the compliance test against the guest virtual machine. This procedure will be repeated with different numbers of compliance tests. First the frequency will be one for every five minutes, and then the frequency will be gradually increased until it reaches the maximum possible.

3.2.3 Experiment 3: Performance LibVMI vs Volatility.

There is expected that the execution time for Volatility on top of LibVMI is higher than running a test directly from LibVMI. To determine if there will be much increase in performance by porting from a Volatilety/LibVMI architecture to a clean LibVMI, there will be interesting to test the potential.

To test this potential there is possible to use some example code provided with LibVMI. One of these codes is to list the running processes, and the same functionality is also provided with volatility. If I measure the

time to execute this code a given number of times for each of the solutions, I will have an indication of the potential performance gain.

3.2.4 Experiment 4: Performance impact with LibVMI.

If the expectation of the pure LibVMI architecture is executing faster than the LibVMI Volatility architecture, and the difference is significant. Then there will be interesting to investigate the maximum frequency that is possible with the pure LibVMI architecture. In addition if the performance impact revealed in Experiment 2: Performance impact on virtual guest section 3.2.2 on page 20 is insignificant, there will be possible to investigate if this architecture will provoke a higher performance impact on the virtual guest.

If this conditions are met the experiment with LibVMI can use the same approach as in Experiment 2: Performance impact on virtual guest. If this experiment will result in an insignificant performance impact on the guest virtual machine, this can be used as evidence to support the theory that VMI techniques can remove some of the performance impact from monitoring agents running on the virtual machine.

Chapter 4

Approach

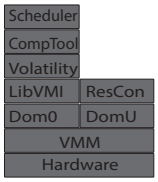
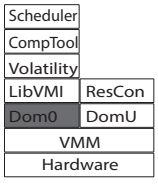


Figure 4.1: The Environment

In this chapter I will explain how I have constructed the environment to support the prototype, and the way it is designed. If you want more detail about the environment than is given in this chapter, the complete procedure is described in Setting up the environment appendix A on page 49. In an attempt to make the description of the environment and the relations between the different modules more understandable I have provided a readers map as illustrated in figure 4.1

4.1 Installing and configuring the dom0

To install Xen Project server with Ubuntu as dom0 server are a bit different that installing Citrix Xen server or VMWare ESX sever. First I had to perform a standard Ubuntu 14.04 server installation, and then I was able to install the Xen 4.4.0 hypervisor. Then after the installation of the hypervisor I could restart the system and boot into the hypervisor. Due to the fair amount of memory (512GB) on the physical server I had some problems on the first boot, but after restricting the dom0 server to only use 4 GB of memory it worked perfectly. This is also considered best practice [34].

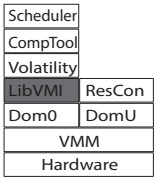


To verify the installation I used the tools provided with Xen to list running virtual machines. In this list the Ubuntu server I had installed was listed as a virtual machine. Before I could start installing virtual machines on Xen I also needed to install and configure a network bridge.

First I deployed some paravirtualized host with the intention of performing the experiments on them. Later in the process I discovered that there were some issues with creating profiles for volatility on paravirtualized hosts, so I also deployed hardware assisted virtual host to

counter this problem. To deploy hardware assisted virtual machines you need a graphical environment, and to install a graphical environment on a dom0 server is not recommended. A solution to this is to export the console to another server or machine with a graphical environment.

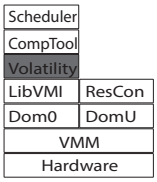
4.2 Installing and configuring libvmi



Before I could install LibVMI I needed to install all the dependencies which are listed in Setting up the environment section A on page 49. After the installation there is some steps and configuration that needs to be done. LibVMI do need the debug symbols which are located in the System.map file located on the /boot directory on the virtual host which are going to be monitored. In addition Libvmi needed some offsets addresses which I collected with running a script on the same virtual host. The offset addresses and the path to the system.map file is gathered in configuration file with a pointer to the name of the virtual guest.

To verify the installation I ran the example code provided by the tool, and I was actually performing VMI. Lbvmi supported the paravirtualized host and I did successfully run the code on PV and HVM machines.

4.3 Installing and configuring Volatility



To use Volatility [33] it is not necessary to do an installation, as long as you download the content from the github repository it is possible to run the python code directly from the Volatility directory. In this way it is possible to uses different versions of volatility from the same machine. Volatility is ordinary design to analyze a memory dump file, and not operate directly on the live memory of a virtual host. In order to make Volatility to work it is necessary to install a plugin named pyvmi [24], which is provided with LibVMI, and copy the pyvmiaddressspace.py to the volatility/plugins/addrspaces/ directory of the volatility installation.

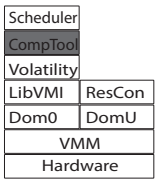
Because the target virtual machine in my environment is an Ubuntu virtual machine, I needed to create a Linux profile for volatility. To create the profile I created a module.dwarf file which contain the structure of the target virtual machine. Then I compressed this file and the system.map of the target virtual machine into a zip file. These procedure did not work for my attempts on paravirtualized virtual machines, but for the hardware assisted machines it worked as expected.

4.4 Creating the prototype. Compliant.pl

The prototype is created in Perl, and the entire script can be located in Compliant script compliant.pl section B on page 61. The prototype main faces is listed below.

- Control the input variable provided by the executer.
- Read the configuration from the configuration file.
- Do the compliant test provided from the configuration file.
- Create and populate a log file if specified.
- Determine and report if the monitored virtual host is compliant.

To list the help use this command:




```
#perl compliant.pl -h
```

Then the help menu is listed.

```
-h for help
-v for verbose ( more output )
-d for debug ( even more output )
-c <filename> for the configuration file
-l <filename> for the log file
```

In the current environment there are many dependencies and things that can go wrong. With this in mind I have provided a verbose and debug mode. A particular debug message that might be handy is the actual volatility command that actually is executed from the prototype.

To make the script run it is mandatory to provide the configuration file. Below I have listed a configuration file example.

Listing 4.1: Example Configuration File

```
1 [vguest4]
2 profile=Linuxubuntu1204x64-3_13_0-32-genericx64
3 eth0=172.24.201.83
4 ps1=apache2
5 kernel=3.13.0-32-generic
```

The first line in this configuration file is the actual name on the target virtual machine. This name needs to be the same name which is given in the configuration file for LibVMI. All of the lines listed after the name in square brackets belong to this host. If you do want to monitor more than one host

you may add another after all the parameters for the first is entered. There is no mandatory order for the rest of the parameters belonging to a host. In addition to the name the prototype need a Linux profile for volatility, and this is given with the profile parameter.

In the third line in the example configuration file the network interface is listed. The prototype expects the network interface to start with “eth“ and a number. The number can vary from 0 to 9. The prototype only supports compliance testing for ipv4. In line five a definition for a complaint test for a running process is listed. This definition starts with “ps“ and a number. It is possible to list more than one process with a maximum of ten (0-9). In the last line the kernel I will test against is given.

When the script is executed, after it has validated parameters, it starts collecting the configuration given in the configuration file. Below I have listed the part form the prototype that contains the collecting configuration part.

```
1  open(CONF, "$CFILE") or die "Error␣opening␣$CFILE␣$!\n
       ↪ ";
2
3  my %guestconf;
4  my $guest;
5  #Read the configuration from the config file.
6  verbose("Read␣the␣configuration␣file␣\n");
7  while ( my $line = <CONF> ) {
8  if ( $line =~ /^\[(\w+)\]/i )
9  {
10 $guest = $1;
11 $guestconf{$guest}{'compliant'} = 'compliant';
12 verbose ("Reading␣config␣guest␣=␣$guest␣\n");
13 }#if
14 elsif ( $line =~ /(^eth\d)=(\d{1,3}\.\d{1,3}\.\d
       ↪ {1,3}\.\d{1,3})/ )
15 {
16 $guestconf{$guest}{$1} = $2;
17 debug("Host␣=␣$guest␣Attribut␣=␣$1␣Value␣=␣$guestconf{
       ↪ $guest}{$1}␣\n");
18 }
19 elsif ( $line =~ /(ps\d)=(\S*)/ )
20 {
21 $guestconf{$guest}{$1} = $2;
22 debug("Host␣=␣$guest␣Attribut␣=␣$1␣Value␣=␣$guestconf{
       ↪ $guest}{$1}␣\n");
23 }
24 elsif ( $line =~ /(kernel)=(\d{1,2}\.\d{1,2}\.\d
       ↪ {1,2}-\d{1,2}-\S*)/ )
```

```
25 {
26 $guestconf{$guest}{$1} = $2;
27 debug("Host = $guest Attribut = $1 Value = $guestconf{
       ↪ $guest}{$1} \n");
28 }
29 elsif ( $line =~ /(profile)=(\S*)/ )
30 {
31 $guestconf{$guest}{$1} = $2;
32 debug("Host = $guest Attribut = $1 Value = $guestconf{
       ↪ $guest}{$1} \n");
33 }
34 }
```

In the start of this part I open the file, and then then each line of the file is read. The data is collected in a two dimensional hash variable, which has this format.

```
{Hostname}{What to test}{Value it should be like}
```

All the values are retrieved from the configuration file with regular expressions, but the expressions do not validate the value is 100 present. This however will probably not make the prototype fail, but the host will in the end be reported as a non-compliant host. To give an example there is possible to create an IP address in the configuration file like 999.999.999.999, and the script will accept it. But finding this configuration in a virtual machine memory is very unlikely and the tested virtual machine will not be compliant.

After the configuration is gathered from the configuration file it is time to do the testing. The two dimensional hash that now contain the configuration is traversed with the help of two while loops.

```
1 foreach my $prguest (keys %guestconf) {
2 foreach my $prattribut (keys $guestconf{$prguest}){
```

Then depending on the "prattribut" variable which contains the information of which test to perform, one of three tests are executed. If prattribut is equal to eth and a number the test for IPaddress are executed. To save some space and enhance the readability I have removed the lines containing feedback messages like debug.

```
1 if ($prattribut =~ /(eth\d)/){
2   my $prnic = $1;
3   open (VCMD, "python vol.py -l vmi://$prguest --
       ↪ profile=$guestconf{$prguest}{'profile'}
       ↪ linux_ifconfig |");
4   while (my $vcmdline = <VCMD>){
```

```
5       #Test for ip on nic
6       if ($vcmdline =~ /(eth\d)\s*(\d{1,3}\.\d{1,3}\.\d
            ↪ {1,3}\.\d{1,3})/){
7         if ($guestconf{$prguest}{$prnic} ne $2){
8           $guestconf{$prguest}{'compliant'} = 'not
                ↪ compliant';
9         }#if
10      }#if ($vcmdline =~ /(eth\d)\s*(\d{1,3}\.\d{1,3}\.\
            ↪ d{1,3}\.\d{1,3})/)
11    }# while (my $vcmdline = <VCMD>)
12    close(VCMD);
13  }#if ($prattribut =~ /(eth\d)/)
```

First the actual NIC that will be tested are collected. According to the configuration file example this will be eth0. Then a Volatility command is executed and the feedback from this command is stored in VCMD. Then the result from this command is traversed line by line to see if there is a line containing the interface I are processing. If the line is found, the prototype tests if the current in memory IP-address matches the desired ip-address from the configuration file. The prototype expects the network interface to be listed in the feedback from Volatility, which should be fixed in a later version.

The next test that is possible is to test for a running process. This time I have also removed the feedback messages to save space.

```
1   #Running test for desierd prosesses.
2   elsif($prattribut =~ /(ps\d)/) {
3     my $prps = $1;
4     my $pscompliant = "not compliant";
5     open (VCMD, "python vol.py -l vmi://$prguest --
          ↪ profile=$guestconf{$prguest}{'profile'}
          ↪ linux_pslist |");
6     while (my $vcmdline = <VCMD>){
7       if ($vcmdline =~ /0x[0-9a-f]*\s(\S*)/){
8         my $prtestvalue = $1;
9         if ($guestconf{$prguest}{$prps} eq $prtestvalue)
              ↪ {
10          $pscompliant = 'compliant';
11        }# if ($guestconf{$prguest}{$prps} eq
              ↪ $prtestvalue)
12      }#if ($vcmdline =~ /0x[0-9a-f]*\s(\S*)/)
13    } #while (my $vcmdline = <VCMD>)
14    close(VCMD);
15    if ($pscompliant eq 'not compliant') #if the prosess
          ↪ was not running the guest is not copliant.{
16      $guestconf{$prguest}{'compliant'} = 'not compliant
            ↪ ';
```

```
17    }#if ($pscompliant eq "not compliant";)
18  }#The prototype expect the network interface to be
      ↪ listed in the feedback from Volatility, which
      ↪ should be fixed in a later version
```

First the process number is collected, before a temporary variable is set to "not compliant". Then the prototype calls Volatility to return all running processes. Then the list is traversed line by line to detect if the running process is running. If the desired process is fount the temporary variable is set to "compliant". Then in the end if the temporary variable is not set to "compliant" the global hash variable is set to "not compliant" for the current host.

In the end I will also test for the desired kernel version. The part of the prototype that is performing this task is listed below without the user feedback messages.

```
1  #Running test for desierd kernel
2  elsif($prattribut =~ /kernel/){
3    while (my $vcmdline = <VCMD>){
4      if ($vcmdline =~ /Linux\sversion\s(\d{1,2}\.\d
         ↪ {1,2}\.\d{1,2}-\d{1,2}-\S*)/){
5        my $pskernelvr = $1;
6        if ($guestconf{$prguest}{'kernel'} ne
           ↪ $pskernelvr){
7          $guestconf{$prguest}{'compliant'} = 'not
             ↪ compliant';
8        }#if ($guestconf{$prguest}{'kernel'} ne
           ↪ $pskernelvr)
9      }#if
10   }#while (my $vcmdline = <VCMD>)
11   close(VCMD);
12 }#elsif($prattribut =~ /(kernel)/)
```

Because it is not possible to run more than one kernel for each host, this test is a bit simpler. The prototype does call for Volatility to return the running kernel version, and then it is compared with the desired kernel version. In most cases Volatility will not be able to run the test if the kernel is changed on the target virtual machine, but if the kernel structure and debug symbols is not changed then it is possible the new kernel version is retuned. In both cases the test will correctly fail and the prototype will report the host as "not compliant".

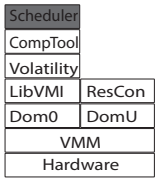
The only thing that is left is to list the compliant status of the hosts that are tested. The code is listed below.

```
1  if ($guestconf{$prguest}{'compliant'} eq 'compliant'){
```

```
2   print 'Guest ',$prguest,' is ',colored ['green
      ↪ on_black'],' compliant',"\n";
3 }
4 else {
5  print 'Guest ',$prguest,' is ',colored ['red on_black
      ↪ '],' not compliant',"\n";
6 }
```

Instead of listing the status of the hosts, it would be possible to change this code to attach or detach the host from the network.

4.5 Creating the scheduler. schedule.pl



To perform the experiments planned for this thesis I needed to create a script that would trigger the prototype with a given frequency and a given number tries. To accomplish this I created the schedule.pl script. If this script is executed with the h parameter this is the result.

```
Usage:
-h for help
-v for verbose ( more output )
-d for debug ( even more output )
-l <filename> for the log file
-n number of loops
-f number of loops/min
```

The important parameters from this script is the f which determine the number of executions that will be done per minute, and the n which determines the total number of executions. The script expects that total number divided by the frequency is a real number. If this is not the case the script will run the command some more until the condition is met. This is better explain if I investigate the script.

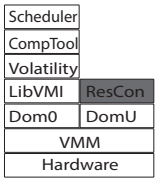
Listing 4.2: schedule.pl

```
1 open(LOG,">$LFILE");
2 my $count =0;
3 my $fcount =0;
4 my $tsleep = 60 / $FLOOP;
5 while ($count++ < $NLOOP) {
6   while ($fcount++ < $FLOOP ){
7     my $ptstart = time;
8     my $pstart = [Time::HiRes::gettimeofday()];
9     system ("perl␣compliant.pl␣-c␣des.cfg");
```

```
10      my $pelapsed = Time::HiRes::tv_interval($pstart);
11      print LOG "PROSESSELAPST = $pelapsed \n";
12      my $ptelapsed = time - $ptstart;
13      sleep ($tsleep - $ptelapsed);
14    #print "$ptelapsed \n";
15    $count++;
16    }#while
17    $fcount=0;
18    $count--;
19  }#while
```

To make sure the tests are evenly distributed I calculate the time it takes to execute the prototype one time. And then the script calls on a sleep routine for the reminding of the time that was allocated for that execution. To give an example: If the frequency is 4 per minute each of the execution have 15 minutes to execute. If the execution only last for 13 seconds then the script will sleep for 2 seconds before the next execution is executed.

4.6 Creating the resource consuming process. rescons.pl



One of the main objectives of the experiments that will be conducted in this thesis is to estimate the performance impact on the target virtual host when a VMI test is conducted from the dom0 machine. In order to obtain data on this performance impact, the idea is to have a resource consuming process executing on the targeted virtual machine, and measure how the performance of this process is changing with different amount of VMI test conducted against it. To perform this test I have created another Perl script. The main part of this script is listed below.

```
1  my $start = [Time::HiRes::gettimeofday()];
2  open(LOG,">$LFILE");
3  my $count =0;
4  while ($count++ < $NLOOP) {
5    my $zipstart = [Time::HiRes::gettimeofday()];
6    system ("zip $ZFILE.zip $ZFILE");
7    my $zipelapsed = Time::HiRes::tv_interval($zipstart)
       ↪ ;
8    print LOG "ZIPELAPST = $zipelapsed \n";
9    system ("rm $ZFILE.zip");
10 }#while
11 my $elapsed = Time::HiRes::tv_interval($start);
12 print LOG "TOTALELAPST = $elapsed \n";
13 close(LOG);
```

The script takes a file as an in parameter, and then the file is compressed and then the compressed file is deleted. This process is repeated a number

of times equal to a number provided as another parameter. To compress a file uses a fair amount of processing power, and will suite our needs sufficiently. I addition to compress and delete the file the script measure the time it takes to compress the file. This measurement is stored in a log file.

Chapter 5

Results

In this chapter I will present the result from the experiments I have conducted. I have collected a large amount of data, so it would be impractical to display the complete collection in this paper. Instead I have extracted and presented the most interesting data related to my analysis and discussion in this chapter. For a dedicated reader the complete data is presented in "Data from the experiments" Appendix E on page 75.

5.1 Experiment 1

In this experiment I have tested the accuracy of my clam that the prototype in the environment I have designed will only consider the in memory configuration. The first test I did in this experiment was to validate that the prototype was reporting the target virtual guest as compliant when it did corresponded correctly with the configuration file. The configuration file used in this experiment is "Example Configuration File" appendix 4.1 on page 25.

After I had validated that a compliant system was reported correctly as a compliant system I started to change the conditions on the targeted virtual machine. First I edited the configuration on the targeted virtual machine, and then I executed a compliant test from the prototype. The next step was to change back the configuration file, before I changed the IP-address in the memory running configuration. To accomplish this I used this command on the target virtual machine.

```
#sudo ifconfig eth0 172.24.201.85
```

In the end I changed back the ip address before the apache2 service was stopped on the target virtual machine. The result of this experiment is listed in the table below.

IP-Address File	IP-Address Running	Apache2 Service	Kernel	result
ok	ok	ok	ok	Compliant
not ok	ok	ok	ok	Compliant
ok	not ok	ok	ok	Not Compliant
ok	ok	not ok	ok	Not Compliant

Table 5.1: Experiment 1: Compliant Result.

5.2 Experiment 2

In this experiment I investigated how the performance was degrading as a result of the test the prototype was conducting against this virtual machine. On the targeted machine I was running the resource consuming process described in "Creating the resource consuming process. rescons.pl" section 4.6 on page 31. First I ran the resource consuming process with no constraints caused by the prototype, in order to create a baseline.

After the baseline was created I did some initial testing and discovered that my prototype used slightly less than 15 seconds to finish a full compliance test. Below is an example result of this testing.

```
1 13.024194
2 12.957368
3 13.028284
4 13.006828
5 12.98066
6 12.983966
7 12.995092
```

Because it was desirable to have evenly distributed tests executed by the scheduler the maximum number of test per minute was set to 4. This resulted in the lode was set from 1 to 4 executions. It is important to remember that one execution of the prototype do result in tree test being performed against the target virtual host. The result is summarized in the table below.

	Base	1/min	2/min	3/min	4/min
Mean	48.17942	47.98267	47.47755	47.46589	47.09673
Standard deviation	0.7525777	0.6214402	0.6041725	0.5885031	0.6203022
Total Time	2411.050474	2401.534411	2375.311718	2374.905427	2356.355373

Table 5.2: Experiment 2: Performance impact on target GVM

To give a visual impression about how the guest virtual machine are affected by the tests that is being performed I have provided a graph representing the mean as the frequency varies.

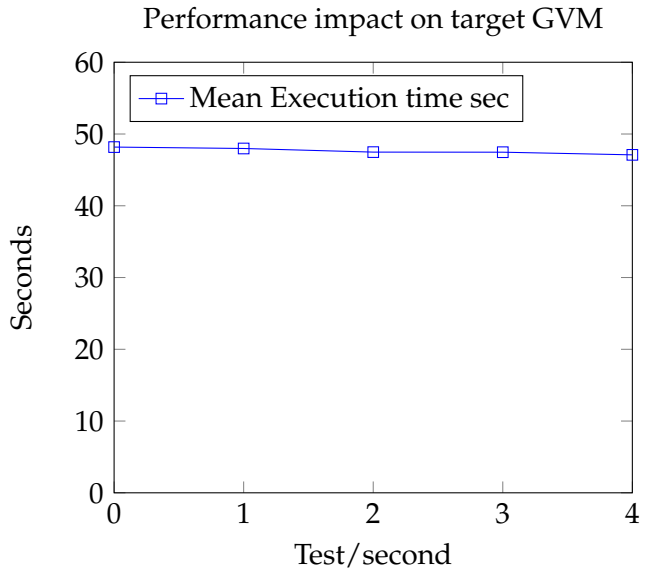


Figure 5.1: Performance impact on target GVM

5.3 Experiment 3

In this experiment I measure how fast a LibVMI and Volatility was able to conduct there test. To measure this I used the scheduler script with some modifications. For this test I did not want any delay, so I commented out the sleep command in line 13 listing 4.2 page 30. In addition I changed the command that should be executed in line 9 with the appropriate command. In the listings below I have extracted some examples from the data collected.

Listing 5.1: Examples Volatility test in seconds.

```
4.797579
4.809241
4.828273
4.836781
4.840397
```

Note that this is one test with Volatility and the prototype uses tree test that are similar.

Listing 5.2: Examples LibVMI test in seconds.

```
0.141318
0.131922
0.133965
0.125096
0.132217
```

In the table below you may see the most important findings. The results are based on 49 executions for each of the methods.

	Volatility	LibVMI
Mean	4.890695	0.1306795
Test/minute	459.1387	12.26819
Standard deviation	0.1555095	0.002283577

Table 5.3: Experiment 3: Result

5.4 Experiment 4

Due to the findings in experiment 3, I continued with experiment 4. In this experiment I am investigating how large a performance degradation it is possible to inflict on the virtual guest by conducting VMI test with Libvmi. One of the intentions is to investigate if the performance degradation is linear or exponential.

To perform the test I did some more modification to the schedule script. To be able to execute more than 60 tests per second I removed the part that spread the test evenly through the minute. The result of this is that some loops of the resource consuming process might be more affected by the LibVMI test than other. I have given a rough illustration on how this will play out in time for a 60 LibVMI tests/minute.

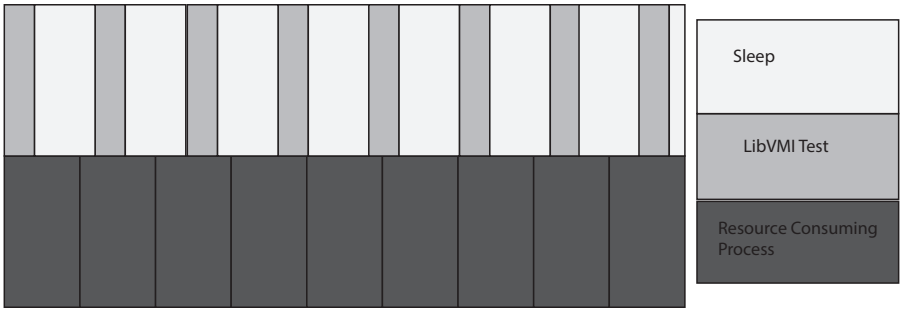


Figure 5.2: 60 LibVMI tests/minute

When I conducted the test I started with 60 LibVMI tests a second, before I increased it with a step of 60 until I reached 360. This was the last step I could have a controlled experiment without future modification of the schedule script. In the table below I have extracted some example data.

For this experiment I first started the scheduler with the LibVMI test, and made sure it would last until the resource consuming process was finished. The resource consuming process was executing 50 times for each of

Base	60/min	120/min	180/min	240/min	300/min	360/min
47.761761	49.846953	57.463015	72.734258	88.73857	103.219395	113.797553
47.06424	50.672137	59.573163	62.943288	71.683805	80.691867	113.440477
47.637454	48.804827	59.836314	71.731202	88.468576	93.175022	111.486272
48.758873	49.536628	60.029929	63.97184	71.433961	89.773232	116.040363
49.03947	49.840646	57.974503	65.159445	86.419854	84.337993	113.915695

Table 5.4: Experiment 4: Performance impact on target GVM

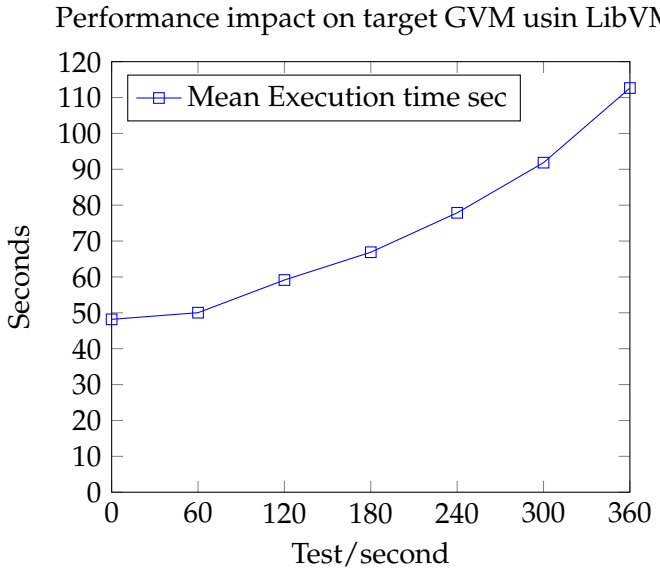


Figure 5.3: Performance impact on target GVM usin LibVMI

the LibVMI frequencies. I have summarized the result in the table below.

	Base	60/min	120/min	180/min	240/min	300/min	360/min
Mean	48.18	50.04	59.14	66.89	77.87	91.85	112.64
Standard deviation	0.753	1.053	0.878	3.162	6.591	6.298	2.162
Total Time	2411.1	2504.0	2959.0	3347.0	3896.2	4595.8	5635.5

Table 5.5: Experiment 4: Summarized performance impact on target GVM

To illustrate the changes in the performance impact on the targeted virtual machine I have created this graph.

Chapter 6

Analysis

6.1 Experiment 1.

In this thesis the most important part was to prove that there is possible to build a tool that uses VMI techniques to determine if a virtual machine has a running configured according to a desired stated. In the first experiment this assumption was tested by validating that it was actually the running configuration that was tested. By validating that it was only when I changed the actual in use IP address and not the configuration file I created evidence that support the claim that it is possible to create a VMI tool that investigates the running configuration. When this is true for the IP-address this will also be true for any configuration existing in memory. This does however not prove it is in impossible for malicious code to camouflage and hide from the VMI tool.

6.2 Experiment 2.

When the evidence for the possibility for a VMI compliance tool was gathered, it was interesting to investigate the performance impact the tool had on the targeted virtual host. It was claimed in the Motivation section 1.1 on page 2 that VMI techniques can free up capacity on the targeted host by removing the traditional agent located on the virtual host. In order for this to be true the impact on the targeted host should be insignificant.

If I investigate the number displayed for Experiment 2 in table 5.2 page 34 I can see the mean of the different experiment is about the same. I can actually see the mean is actually decreasing, which also is illustrated in graph 5.1. Up front I expected a slightly increase in mean as the frequency of test conducted against the targeted virtual host. When the mean is decreasing instead this suggests that there are other factors that are more dominate for the local performance on the virtual host than the VMI tests.

If I investigate the Standard deviation it is not very large related to the mean. This does indicate that most of the data that was collected is not

far from the mean. Which does indicate there has not been a large variance in the load on the targeted virtual machine. With this observation I can assume that the unknown factors discussed with the decreasing mean in the previous paragraph are small as well. I can argue that this count as evidence supporting the clam of a light performance impact on the targeted virtual host.

6.3 Experiment 3.

In experiment 3 I assumed that the dom0 execution was faster if I conducted test only using LibVMI instead of LibVMI Volatility architecture. With this assumption there was interesting to investigate potentially how much faster the change of architecture could be. If I use the mean execution time for both architectures I can calculate the difference.

$\frac{mean(volatility)}{mean(LibVMI)} = \frac{4.890695}{0.1306795} = 37.43$

This is a rather significate difference, and this result supports as a good evidence that LibVMI is faster than Volatilety/LibVMI.

6.4 Experiment 4.

In this experiment the intention was to investigate the possibility to provoke significant performance degradation on the target virtual machine by exposing it to a high frequency of LibVMI tests. In addition it was interesting to see if there was a breaking point in the performance degradation at a specific frequency.

In this experiment the result was more as expected up front, like the mean execution time is increasing as the LibVMI test frequency increase as displayed in table 5.5 on page 37. But if I investigate the graph 5.3 on page 37 I can see there is no obvious breaking point in the graph. The rate the graph is growing is increasing as the frequency getting closer to the maximum, but it is interesting that a clear breaking point is not there.

The absence of a breaking point can be explained with how the LibVMI interacts with the targeted virtual machine. When LibVMI retrieves the information it needs from the targeted virtual machine it actually pauses the virtual machine to get a concise extraction of the part of the memory it needs. After this information is extracted the virtual guest is resumed, and there has actually not happened anything on the targeted virtual machine except it has lost some time. To clarify some ground for misinterpretation, the clock on the virtual machine is not a virtual clock but a representation of the host physical clock.

Given the interpolation about the pausing and resuming of the targeted virtual host does not explain way the graph is curved. To explain this I may assume there is some overhead that is actually affecting the virtual machine in addition to the lost time. Given the data that is collected in this experiment, there is an indication that there will not be a breaking point even if the experimented was supplemented with data from even higher frequencies.

Another possible observation is the great variance in the standard deviation given in table 5.5. There is hard to explain conclusively why this is happening, but one explanation might be the distribution in time of the test illustrated in 5.2 36. On a low frequency the test window is in most cases only affecting one resource consuming process. When the frequency is increasing there is more variation on how many resource consuming processes are affected by the test window and the standard deviation is increasing. In the highest frequencies the sleep window is so small that the LibVMI tests do affect the resource consuming process in more equal degree. Based on this assumption being correct the standard deviation would be kept lover for all frequencies if the LibVMI test was evenly distributed true the experiment.

Chapter 7

Discussion and Future Work

In this chapter the different aspects of this thesis will be discussed, before the conclusion and future work is presented.

7.1 Retrieving the data

In this thesis most of the important data is collected by a tool that specifically was developed specifically for this project. Free tools are available that would solve the task satisfyingly, but when the tool was created in house it was easy to get control on how the data was collected and presented. The main purpose was to create a large workload and measure the execution time. The workload as described was created by compressing and deleting the newly compressed file a given number of times. This approach is somewhat simple, but given the desired date, the result should be sufficient.

There are however some uncertainties in this approach, as it is for many systems with shared resources. One of these uncertainties is that the virtual disk, the virtual machine used is located on the same physical disk as the dom0 server are using. A consequence of this architecture is that disk activity on the dom0 server can impact the reading and writing speed on the targeted virtual machine. Because the targeted virtual machine need to read the file before it can compress it, and then it will write the compressed file back to the disk. Simultaneously the test that was executed on the dom0 machine did created a log which was written to the same physical disk. However all the tests done in this thesis were executed in a strictly controlled environment with no other humanly initiated task being executed. This controlled environment will help ensure that the influence caused by the dom0 activities to the read and write activities on the targeted virtual machine is about the same for the entire experiment. This is ok because the part I wanted to investigate was how the performance on the targeted machine was influenced by VMI tests. Another factor that protects the quality of the results is that the file being compressed was significantly larger than the log file on the dom0 machine. The compressed file was about 1 GB while the dom0 log file was less than 1KB.

In addition to the issues described with the sheared resources, there are also some uncertainties with the start time on for the processes on dom0 and targeted virtual machine. To make sure the process on the targeted virtual machine always was exposed to the activities from the VMI process on the dom0, the dom0 VMI process was started first and made sure it lasted longer than the process on the targeted virtual machine. Because the process was manually started there was some room for uneven execution. But given the large number of tests done and the small standard deviation the data collected can be considered reliable.

7.2 The collected data

If the thesis had lasted for a longer period of time, there would be interesting to collect a wider selection of data. Especially for the unexpected result in "Experiment 2" section 5.2 page 34, where the mean execution time decreased instead of increasing. For instance if the disk waits and data rate transfer was collected there would be possible to investigate if the strange result was caused by disk issues. In addition the same types of data could be collected for the CPU like CPU wait and CPU load, to see if the explanation was CPU related. It would also be interesting to run this entire experiment one more time and see if the resulted reoccurred, or if the next test revealed a more horizontal or slightly increasing graph.

The data retrieved is sufficient to conclude if there was a significant performance impact on the targeted virtual machine, but if this additional data was collected this might be used to design a more efficient VMI tool.

If the experiment was reproduced it would be expected to have some sort of similar result. This would of course depend on the hardware that would host the environment, but the large lines should be about the same. An exception to this is as partly discussed the unexpected result in experiment 2, which might be closer to what was expected.

7.3 The construction of the prototype.

One of the challenges in this thesis was to find the proper tools to be used to construct the prototype. Evan though the field of VMI is a rather new research area, there has been a considerable amount of research on the topic. This research has resulted in large amount of tools available, and to find proper tools with the desired maturity required a large amount of background research.

When the tool was selected the model did not look too complicated, but it turned out to have a lot of dependencies required to make it work. One

of the largest obstacles in this thesis was the lack of support for paravirtualized host for Volatility. This was not easily obtainable information, and this combined with the fact that almost all problems related with making Volatility work is caused by incorrectly created profiles, resulted in a significant amount of effort being used to solve the issue. This issue is now being addressed by the Volatility development team and is considered a work in progress.

After the environment was created, the construction of the prototype was more according to what was expected, and required average scripting skills. The largest division from the initial plan was the decision to not include a capability to connect and disconnect the virtual machines from the network and instead report on the compliance status. This was due to the practical concerns about the experiments that were going to be executed, and an instruction on how this should be developed was considered an acceptable tradeoff. In addition there was not desired to modify the script after the experiments were executed so uphold the integrity of the experiments data by the scripts being represented in their original form.

7.4 The Selected Approach

The approach selected in this thesis is so far the only approach available to answer the questions that are asked. An un-mentioned but strong characteristic of this thesis is the use of open software, which is the case for all the software that has been used for this project. The same solution should be possible to create with other virtualization platforms and tools, like VMware and there vshield solution [32].

Another solution that might have solved the problem statement, and even with fewer difficulties, would be to use a Microsoft window operating system on the virtual host. If this approach had been selected there would be no need to create the Volatility profiles, and probably saved some development time. In addition it would be possible to use Citrix Xen server instead of the Ubuntu dom0. With this solution there would be more management tools available out of the box, but the drawback would be the hardened dom0 server. The hardened Dom0 is more secure, but it is more challenging to install additional software like LibVMI with its dependencies. And most likely with the additional software installed, the solution would probably not be supported by Citrix.

With some changes to the problem statement the approach would change drastically. The most obvious change that would change the approach considerably is if I should not use VMI. Then the model would be based on interacting with the VMs using the network. This would also most likely result in the use of other tools and would result in other challenges and solutions.

7.5 Repeat the project.

If it became relevant to redo the project it would probably be possible to do without the use of considerable resources. First of all the software is freely available as open source. In addition all the steps are documented in Setting up the environment appendix A page 49. There should be possible to replicate the whole environment only by cutting and pasting the command listen in this appendix. There might be necessary to correct some of the commands, and the installations of the operation system are not documented. In addition to the instruction all the scripts is also listed in appendix B, C and D.

Looking at the approach in retrospect there is some changes that could be done in order to increase the quality of the project. First of all if a new version of Volatility is available that supports paravirtualized machines that should be used to expand the supported hosts. In addition to the new support there could be extracted more parameters from the experiments as described earlier in the discussion.

While the performance impact on the targeted virtual machine has been investigated, the performance impact on the Dom0 server has not been devoted much attention. Potentially if the Dom0 machine is taxed too much the result can be that the whole infrastructure will be suffering. It was not necessary to collect this data to answer the problem statement, but the data collected would be a great asset to justify future research to overcome this challenge.

7.6 Relation to Existing Work

As mention earlier there has been conducted a considerable amount of research on the VMI problem domain and most of this work has been exploratory research on how to develop deferent security tools. But in most of the cases the security tool does try to detect or protect against some form of malicious code. What is relatively unique about the tool developed in this thesis is that it does not only address the danger about this malicious code. I addition it intends to protect against well-meaning system administrators doing configurations without following the correct process.

The tool created in this thesis can actually be illustrated as the first step into creating a datacenter “immune system“. Imagine the way as the white blood cells isolate and encapsulate foreign particles or cell in the body, before it kills it or pushes it out. In this same way this prototype are making way for a computer white cell that can isolate and either kill or reset the server with a foreign object (not compliant configuration). An important

notification is that this diagnostic can be done before the server is even allowed into the network. Just like the body's immune system protect the body form foreign objects even entering the body.

In addition to being an immune system the prototype also frees up resources on the targeted machine because with this design there is no need for an agent running locally on the virtual machine. There has been some research on how the targeted systems are affected, but in most cases this has been left out it is the execution of the VMI tool that has been investigated. This thesis complements this part of the research..

7.7 The Intended Consumer

The main target user for this prototype was the system administrator of a virtual infrastructure. This also includes the system administrators for cloud providers, which provide services as rely on a specific configuration. But there are other potential consumers as well. This tool given it is being developed future, can be used by process manages responsible for the configuration process. If a configuration manager can report on the state of the datacenter with only a few minute delay this can be of great value. And this can be done even if the virtual machine is not connected to the network.

7.8 Conclusion

In this thesis a prototype that uses VMI techniques to investigate if targeted virtual machines are compliant according to a predefined configuration stored in a more secure location. The tool has these capabilities.

- Control the input variable provided by the executer.
- Read the configuration from the configuration file.
- Do the compliant test provided from the configuration file.
- Create and populate a log file if specified.
- Determine and report if the monitored virtual host is compliant.

In addition the tools necessary to measure the performance impact on the targeted virtual machine was created.

The performance impact on the targeted virtual machine was investigated. The conclusion from the data collected was that the performance impact on the targeted virtual machine was insignificant if the test frequency was small.

The result from the experiment also revealed that there is a great performance potential in developing the tool with only LIbVMI instead of using Volatility on top of LiBVMI. In the experiment conducted the pure LibVMI architecture was 37,43 times faster.
The final discovery was that there is possible to provoke significant performance degradation by using a large test frequency.

7.9 Future work

Given the result form the experiment there would be great potential in developing a similar tool like the on developed in this thesis, but based on a pure LibVMI architecture. This would significantly reduce the resource taxation of the dom0 server.

The fact that the prototype developed in this thesis is running on the dom0 server raises some concerns. It would be a large problem if the load on the dom0 server would be too large because this may result in the whole virtual infrastructure suffering. To counteract this, it can be investigated the possibility to move the VMI tool from the dom0 to another virtual server. In theory there should be possible to delegate the necessary privileges to a domU server, but with this approach the security aspect is essential. In the newest version of Xen it comes with the support of deploying a Secure Domain server instead. This server can be used to host VMI security applications, and could be used as an approach.

To future enhance the capabilities of the prototype to be a true datacenter immune system there should be developed more functionality. In many cases a server can be reverted back to a specific state in time, without losing any data. An example is a web server connected to a database. If all the data are saved in the database, and nothing on the web server, it is safe to revert the web server into it initial state. To future develop the prototype with a capability to revert a server with a drifting configuration back to a desired state, this could be proven to be valuable.

Appendix A

Setting up the environment

A.1 Installation Dom0

For my environment I installation Ubuntu 14.04 on a Dell PowerEdge r810 with 2 x 12 cores and 512GB RAM. First I installed the xen hypervisor by using this command.

```
# sudo apt-get install xen-hypervisor-amd64
```

There is considered best practice to dedicate a fixed amount of memory to the Dom0 server, and to do this you will need to edit the grub loader. To do this in Ubuntu you can edit /etc/default/grub with your favorite editor. I used nano.

```
# sudo nano /etc/default/grub
```

And I added

```
# GRUB_CMDLINE_XEN_DEFAULT="dom0_mem=min:4096M,max
    ↪ :4096M"
```

In addition you need to install this packet.

```
#sudo apt-get install gedit
```

To make Ubuntu update the grub loader you need to run this command.

```
sudo update-grub
```

If you are using an older version of Ubuntu than 14.04 you also need to edit the grub loader so it boots into the hypervisor as default. But because I have used this version I are now ready to boot the server by running this command.

```
# sudo reboot
```

To be able to use hardware assisted virtualization the vt-x or ADM-v need to be enabled in the bios, and this boot might be a good time to ensure this. After the boot the VMM are running and the Dom0 is running beside the VMM. Now it is time to install the libvmi [14] tool. As documented on the project web site there are some dependencies, but in my experience I do need some more packages are needed as well. First of all I need autoconf and make to be able to install packages that are not distributed with apt-get. To do this I ran these commands.

```
# sudo apt-get install autoconf
#sudo apt-get install make
```

For the first dependency libxc that also need libtool run these commands.

```
#sudo apt-get install libtool
#apt-get install libxc-dev
```

In addition I installed bison and flex.

```
#sudo apt-get install bison
#sudo apt-get install flex
```

In addition I experienced that this installations was needed.

```
# sudo apt-get install gfortran
#sudo apt-get install libxen-dev
```

According to the project webpage libxenstore is also needed, but this was installed together with the hypervisor. So now I are ready to install the libvmi. First I download the version I would like, and unpack it.

```
#Sudo wget https://code.google.com/p/vmitools/
    ↪ downloads/detail?name=libvmi-0.10.1.tar.gz&can
    ↪ =2&q=
tar -xf libvmi-0.10.1.tar.gz
```

Then I enter the libvmi directory and run autogen and configure and make.

```
#cd libvmi-0.10.1
./autogen.sh
./configure
make
```

Then I install libvmi by running this commands.

```
make install
ldconfig
```

In addition I need to create a configuration file that contains the information libvmi need to bridge the semantic gap to do VM inspection. How to do this is explained in section Configure Libvmi section A.5 on page 55.

A.2 Install and configure the network bridge

For the virtual machines to be able to connect to the network I need to set up and configure a network bridge. First I need to log in on the Dom0 server and then I install the bridge-utils packet by running this command.

```
#sudo apt-get install bridge-utils
```

In the case the Dom0 computer is a desktop installation, which is not recommended, the Network Manager need to be disabled by running these two commands.

```
# sudo update-rc.d network-manager disable
# sudo /etc/init.d/network-manager stop
```

Then I are ready to configure the network bridge by editing this file /etc/network/interfaces. I ran this command.

```
# sudo nano /etc/network/interfaces
```

My file looked like this.

```
# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto em1
iface em1 inet manual

auto xenbr0
iface xenbr0 inet static
        address 172.24.201.138
        netmask 255.255.255.0
        gateway 172.24.201.1
        bridge_ports em1
        dns-nameservers 172.24.201.51 172.24.201.50
```

Then I need to take down and up the em1 networking interface and enable the xenbr0 by running this command.

```
# sudo ifdown em1 && sudo ifup xenbr0 && sudo ifup em1
```

Note to restart the actual networking service is no longer supported in Ubuntu 14.04.

A.3 Install a Paravirtualized guests

There is many methods to deploy virtual machines on Xen, but in my case I selected do build them manually. First I ran this command to find the volume groupe.

```
# sudo pvs
```

In my case I got this result.

```
  PV         VG           Fmt  Attr PSize   PFree
  /dev/sda5  s02tvm01-vg  lvm2 a--  135.26g 106.64g
```

From this I find that s02tvm01-vg is my volume group. Then I can create the LVM volume for the new VM by running this command.

```
# sudo lvcreate -L 4G -n vguest1 /dev/s02tvm01-vg
```

Then I created a directory and downloaded the necessary files from an archive mirror [20], by running this commands.

```
sudo mkdir -p /var/lib/xen/images/netboot/ubuntu/
cd /var/lib/xen/images/netboot/Ubuntu
# sudo wget http://ftp.uninett.no/ubuntu/ubuntu/dists/
    ↪ precise/main/installer-amd64/current/images/
    ↪ netboot/xen/initrd.gz
# sudo wget http://ftp.uninett.no/ubuntu//ubuntu/dists
    ↪ /precise/main/installer-amd64/current/images/
    ↪ netboot/xen/vmlinuz
```

Then I need to create a config file for the virtual machine /etc/xen/vguest1.cfg

```
name = "vguest1"
memory = 1024
disk = ['phy:/dev/s02tvm01-vg/vguest1,xvda,w']
```

```
vif = [ 'bridge=xenbr0' ]
kernel = "/var/lib/xen/images/netboot/ubuntu/vmlinuz"
ramdisk = "/var/lib/xen/images/netboot/ubuntu/initrd.
    ↪ gz"
extra = "debian-installer/exit/always_halt=true␣--␣
    ↪ console=hvc0"
```

Now I are ready to start and install the guest operating system by using this command.

```
sudo xl create /etc/xen/vguest1.cfg -c
```

The c switch is used to connect to the console. After the installation the vm is powered off, and I can set the pygrub as the bootloader by using this command.

```
sudo ln -s /usr/lib/xen-4.1/bin/pygrub /usr/bin/pygrub
```

Then I need to do some changes to the cfg file I created. My /etc/xen/vguest1.cfg file locked like this.

```
name = "vguest1"
memory = 1024
disk = ['phy:/dev/s02tvm01-vg/vguest1,xvda,w']
vif = [ 'bridge=xenbr0' ]
bootloader = "pygrub"
#kernel = "/var/lib/xen/images/netboot/ubuntu/vmlinuz"
#ramdisk = "/var/lib/xen/images/netboot/ubuntu/initrd.
    ↪ gz"
#extra = "debian-installer/exit/always_halt=true --
    ↪ console=hvc0"
```

To start the vm you use this command

```
# sudo xl create /etc/xen/vguest1.cfg -c
```

I am using putty to connect with ssh to my dom0 server, and then from the console to vguest1 and back to the dom0 server I uses the <ctr>+5 to exit.

A.4 Install hardware assisted virtual host

Another method to install a host in a Xen environment is hardware assisted virtual host or a hardware virtual machine (HVM) which it is named when I run on Xen. To be able to deploy hardware assisted virtual machines the physical CPU needs to support this technology and it need to be enabled in BIOS. The approach to deploy a HVM is similar to PV guest. If I will use a LVM volume for the disk I need to determine the LVM volume group.

```
# sudo pvs
```

In my case I got this result.

```
PV VG Fmt Attr PSize PFree
/dev/sda5 s02tvm01-vg lvm2 a-- 135.26g 106.64g
```

From this I find that s02tvm01-vg is my volume group. Then I can create the LVM volume for the new vm by running this command.

```
# sudo lvcreate -L 10G -n vgues41 /dev/s02tvm01-vg
```

Then I created a directory and downloaded the iso file for Ubuntu Server 14.04.1. by running this commands.

```
sudo mkdir -p /var/lib/xen/images/iso
cd /var/lib/xen/images/iso
# sudo wget http://releases.ubuntu.com/14.04.1/ubuntu
    ↪ -14.04.1-server-amd64.iso
```

Then I need to create a config file for the virtual machine /etc/xen/vguest4.cfg

```
builder = "hvm"
name = "vguest4"
memory = "1024"
vcpus = 1
vif = [ 'bridge=xenbr0' ]
disk = ['phy:/dev/s02tvm01-vg/vguest4,hda,w','file:/
    ↪ var/lib/xen/images/iso /ubuntu-14.04.1-server-
    ↪ amd64.iso,hdc:cdrom,r']
vnc = 1
vncconsole=1
boot="dc"
stdvga = 0
```

Now I are ready to start and install the guest operating system by using this command.

```
sudo xl create /etc/xen/vguest4.cfg
sudo vncviewer localhost:0
```

The last command is used to connect to the console, but this does need a graphical environment. A graphical environment is not always the case of a dom0 server. In theory you could connect to the virtual host with VNC, but for security reasons vnc on xen only listens on the loopback interface

127.0.0.0. There are different ways to work around this problem. One of them is to export the display to a computer with graphical environment. To do these open a terminal on the computer with a graphical environment and enter the command under.

```
# xhost +
```

On the dom0 Server you enter these commands.

```
# DISPLAY=<ipaddress>
#export DISPLAY
#sudo vncviewer localhost:0
```

The ip address is the address of the host with a graphical environment. After the installation you should power off the new virtual guest and edit the /etc/xen/vguest4.cfg file to remove the cd. Mine looked like this.

```
builder = "hvm"
name = "vguest4"
memory = "1024"
vcpus = 1
#vif = ['']
vif = [ 'bridge=xenbr0' ]
#disk = ['phy:/dev/s02tvm01-vg/vguest4,hda,w','file:/
    ↪ home/admanl/iso/ubuntu-14.04.1-server-amd64.iso,
    ↪ hdc:cdrom,r']
disk = ['phy:/dev/s02tvm01-vg/vguest4,hda,w']
vnc = 1
vncconsole=1
boot="dc"
```

To start the vm you use this command

```
# sudo xl create /etc/xen/vguest4.cfg
```

A.5 Configure Libvmi

After the guest virtual machine in installed there is time to configure libvmi. In order for libvmi to find the guest virtual machine and be able to introspect it, I need to create a configuration file named /etc/libvmi.conf. To populate this file you will first need the virtual machine name that was given during the creation of the virtual machine. To fine this name you can run this command.

```
# sudo xl list
```

My result was.

```
Name                                        ID    Mem
    ↪ VCPUs      State    Time(s)
Domain-0                                     0   4093
    ↪     24     r-----    2373.5
ubuntu                                       2   1024
    ↪      1     -b----     118.5
vguest1                                      7   1024
    ↪      1     -b----      84.5
```

In addition for a linux machine you need to find [14]

- sysmap = this is the path to the guest virtual machine which you need to copy from the guest virtual machine to some place on the Dom0 server.
- ostype = “Linux“ or “Windows“
- `linux_mm = Is the Offset to the task_struct->mm.`
- `linux_pid = Is the Offset to the task_struct->pid.`
- `Linux_pgd = is the offset to the task_struct->pgd.`

To collect this information libvmi has provided a tool that can be copied from`$Home`/libvmi-0.10.1/tools/linux-offset-finder/ to the guest virtual machine. When you have copied the tool to the guest virtual machine you first run.

```
# sudo make
```

Then you will get a findoffsets.ko file and you can run this command.

```
# sudo insmod findoffsets.ko
```

You will then find the necessary information in `$dmesg` or /var/log/syslog. Example from my syslog.

```
Nov  5 16:07:46 vguest1 kernel: [ 3953.223721] Module
    ↪ FindOffsets loaded.
Nov  5 16:07:46 vguest1 kernel: [ 3953.223725]
Nov  5 16:07:46 vguest1 kernel: [ 3953.223744] [domain
    ↪  name] {
Nov  5 16:07:46 vguest1 kernel: [ 3953.223749]
    ↪ ostype = "Linux";
Nov  5 16:07:46 vguest1 kernel: [ 3953.223755]
    ↪ sysmap = "[insert␣path␣here]";
Nov  5 16:07:46 vguest1 kernel: [ 3953.223763]
    ↪ linux_name = 0x470;
```

```
Nov  5 16:07:46 vguest1 kernel: [ 3953.223769]
    ↪ linux_tasks = 0x248;
Nov  5 16:07:46 vguest1 kernel: [ 3953.223775]
    ↪ linux_mm = 0x280;
Nov  5 16:07:46 vguest1 kernel: [ 3953.223781]
    ↪ linux_pid = 0x2bc;
Nov  5 16:07:46 vguest1 kernel: [ 3953.223787]
    ↪ linux_pgd = 0x58;
Nov  5 16:07:46 vguest1 kernel: [ 3953.223792] }
```

The resulting /etc/libvmi.conf looked like this.

```
vguest1 {
      ostype = "Linux";
      sysmap = "[/etc/vm/System.map-3.2.0-70-generic]";
      linux_name = 0x470;
      linux_tasks = 0x248;
      linux_mm = 0x280;
      linux_pid = 0x2bc;
      linux_pgd = 0x58;
}
```

A.6 Install Volatility and dependencies

To install volatility [33] I first need to install the dependencies and some useful tools. To do this first run this command.

```
sudo apt-get install subversion pcregrep libpcre++-dev
    ↪  python-dev make git zip unzip -y
```

Then I need to install PyCrypto.

```
sudo wget https://ftp.dlitz.net/pub/dlitz/crypto/
    ↪ pycrypto/pycrypto-2.6.1.tar.gz
tar -zxvf pycrypto-2.6.1.tar.gz
cd pycrypto-2.6.1/
sudo python setup.py build
sudo python setup.py build install
cd ..
```

Then I install Distrom

```
sudo wget https://distorm.googlecode.com/files/
    ↪ distorm3.zip
sudo apt-get install unzip -y
sudo unzip distorm3.zip
```

```
cd distorm3/
sudo python setup.py build
sudo python setup.py build install
cd ..
```

Then I install Yara.

```
sudo wget http://yara-project.googlecode.com/files/
    ↪ yara-1.4.tar.gz
sudo tar -zxvf yara-1.4.tar.gz
cd yara-1.4/
sudo ./configure
sudo make
sudo make install
cd ..
```

Then I install Yara Python.

```
wget https://yara-project.googlecode.com/files/yara-
    ↪ python-1.4a.tar.gz
sudo tar -zxvf yara-python-1.4a.tar.gz
cd yara-python-1.4a/
sudo python setup.py build
sudo python setup.py build install
cd ..
```

And I need to edit the file /etc/ld.so.conf

```
include /etc/ld.so.conf.d/*.conf
''/usr/local/lib''
```

And in the end I get a copy of the volatility repository from github.

```
git clone https://github.com/volatilityfoundation/
    ↪ volatility.git
```

For the uses I need in this thesis there is no need for actually install volatility, I only need the directory with a copy of volatility. If I do not install volatility then it is possible to use different versions of the product simultaneously, just cep the different versions in different folders. To use the different versions just run the vol.py file from the desired versions directory.

A.6.1 Create a Profile for Volatility

To be able to analyze a memory dump from a Linux system with Volatility there is necessary to create a Linux profile for that particular system. A profile is a zipped file that contains to files, on with the debugging symbols and one with the kernels data structure. This information is then used by

volatility to locate and parse the desired information from the memory. It is necessary for Volatility to work properly that the profile is created on a system with the same Linux version and kernel version as the system that will be analyzed. For this thesis the profile need to be created on the virtual guest. (vguest4). To create this profile I fist need to install dwarfdump by using this command.

```
# sudo apt-get install dwarfdump -y
```

And I need GCC/make.

```
#sudo apt-get install build-essential -y
```

Then I need to find the kernel version by using this command.

```
1 #sudo uname -a
```

Then I have the needed information to install the headers for building kernel modules. This is in my experience usually installed, but to be sure it should be apt-get installed. And be specific. This is a general command that should work in most cases.

```
#sudo apt-get install linux-headers-$(uname -r) -y
```

In addition to this volatility need to be downloaded to this computer as well.

```
git clone https://github.com/volatilityfoundation/
    ↪ volatility.git
```

Now I will create vtypes or a file that contains the kernels data structure. To do this I only need to run a make command in the tools/linux directory. This sometimes does fail if I are not the owner of this directory. It is also possible it will fail if I use sudo to run the make command. This commands should result in a successful creation of a module.dwarf file.

```
#sudo chown <user> volatility/tools/linux
#cd volatility/tools/linux
#make
```

The debug symbols are percent in the System.map file located in the /boot folder. The System.map file and the module.dwarf file should then be zipped into one file and copied into the volatility/volatility/plugins/overlays/ directory. This command should work with some editing on most Ubuntu systems.

```
#sudo zip ~/volatility/volatility/plugins/overlays/
    ↪ linux/
ubuntu1204x64-$(uname -r).zip ~/volatility/tools/linux
    ↪ /module.dwarf /boot/System.map-$(uname -r)
```

For the thesis this file needed to be copied to the volatility/volatility/plugins/overlays/ directory on the dom0 server. Example command.

```
#sudo scp ~/volatility/volatility/plugins/overlays/
    ↪ linux/
ubuntu1204x64-$(uname -r).zip <user>@/home/<user>/
    ↪ volatility/volatility/plugins/overlays/linux/.
```

To verify that the profile is present in the system you can run this command.

```
# sudo python vol.py --info | grep Linux
```

Then al the linux profiles will be listed.

A.7 Install PyVMI

To make volatility able to read the memory provided by libvmi I need to install a plugin named PyVMI [24]. The plugin is downloaded as a part of libvmi, and is located in /libvmi-0.10.1/tools/pyvmi directory. Enter the directory and install the plugin by using these commands.

```
#cd /libvmi-0.10.1/tools/pyvmi
#sudo python setup.py build
#sudo python setup.py install
```

When this is done you only need to copy the pyvmiaddressspace.py into the volatility/plugins/addrspaces/ directory.

```
#sudo cp pyvmiaddressspace.py volatility/plugins/
    ↪ addrspaces/.
```

Now it is ready to start introspecting the virtual guest by using volatility. The syntax is like this.

```
python vol.py -l vmi://<VirtualGuest> --profile=<
    ↪ profileforvirtualguest> <command>
```

Appendix B

Compliant script compliant.pl

```
1  #!/usr/bin/perl
2
3  # our needed packages
4  use strict "vars";
5  use Getopt::Std;
6  use Term::ANSIColor;
7  use Time::HiRes;
8  # Global variables
9  my $VERBOSE = 0;
10 my $DEBUG = 0;
11 my $LFILE;
12
13 # commandline options
14
15 my $opt_string = "vdhc:l:"; # print out help message
16 getopts( "$opt_string", \my %opt ) or usage() and exit
       ↪ (1); #or die "Read the manual\n";
17
18 $VERBOSE = 1 if $opt{'v'};
19 $DEBUG = 1 if $opt{'d'};
20 if ( $opt{'h'} ){
21      usage();
22      exit 0;
23 }
24
25 my $CFILE = $opt{'c'};
26 debug("Config file was $CFILE\n");
27
28 ( usage() and die "Please supply a config file name\n"
       ↪ ) unless stat($CFILE);
29 if ( $opt{'l'} ){
30          $LFILE = $opt{'l'};
31          debug("Logfile file was $LFILE\n");
32 }
```

```
33
34 ####################
35 #     Main part      #
36 ####################
37 my $start = [Time::HiRes::gettimeofday()];
38 if ( $opt{'l'} ){
39         open (LOG, ">>$LFILE")or die "Error opening
           ↪ $LFILE $!\n";
40 }
41 open(CONF, "$CFILE") or die "Error opening $CFILE $!\n
     ↪ ";
42
43 my %guestconf;
44 my $guest;
45 #Read the configuration from the config file.
46 verbose("Read the configuration file \n");
47 while ( my $line  = <CONF> ) {
48  if ( $line =~ /^\[(\w+)\]/i )
49  {
50  $guest = $1;
51  $guestconf{$guest}{'compliant'} = 'compliant';
52  verbose ("Reading config guest = $guest \n");
53  }#if
54  elsif ( $line =~ /(^eth\d)=(\d{1,3}\.\d{1,3}\.\d
       ↪ {1,3}\.\d{1,3})/ )
55   {
56   $guestconf{$guest}{$1} = $2;
57   debug("Host = $guest Attribut = $1 Value =
       ↪ $guestconf{$guest}{$1} \n");
58   }
59   elsif ( $line =~ /(ps\d)=(\S*)/ )
60   {
61   $guestconf{$guest}{$1} = $2;
62   debug("Host = $guest Attribut = $1 Value =
       ↪ $guestconf{$guest}{$1} \n");
63   }
64   elsif ( $line =~ /(kernel)=(\d{1,2}\.\d{1,2}\.\d
       ↪ {1,2}-\d{1,2}-\S*)/ )
65   {
66   $guestconf{$guest}{$1} = $2;
67   debug("Host = $guest Attribut = $1 Value =
       ↪ $guestconf{$guest}{$1} \n");
68   }
69   elsif ( $line =~ /(profile)=(\S*)/ )
70   {
71   $guestconf{$guest}{$1} = $2;
72   debug("Host = $guest Attribut = $1 Value =
       ↪ $guestconf{$guest}{$1} \n");
```

```
73    }
74  }
75
76
77  foreach my $prguest (keys %guestconf) {
78   foreach my $prattribut (keys $guestconf{$prguest})
79   {
80   debug("Prosessing␣Hist=$prguest␣Attribut=$prattribut␣
        ↪ Value=$guestconf{$prguest}{$prattribut}␣\n");
81   #Running test for desierd network configuration.
82   if ($prattribut =~ /(eth\d)/)
83    {
84    my $prnic = $1;
85    debug("Prosessing␣nic␣=␣$prnic␣\n");
86    debug("Running␣command␣python␣vol.py␣-l␣vmi://
        ↪ $prguest␣--$guestconf{$prguest}{'profile'}␣
        ↪ linux_ifconfig␣|\n");
87    print "INFO␣:␣Framework␣used␣to␣run␣test␣=␣";
88    open (VCMD, "python␣vol.py␣-l␣vmi://$prguest␣--
        ↪ profile=$guestconf{$prguest}{'profile'}␣
        ↪ linux_ifconfig␣|");
89
90    while (my $vcmdline = <VCMD>)
91     {
92     #Test for ip on nic
93     if ($vcmdline =~ /(eth\d)\s*(\d{1,3}\.\d{1,3}\.\d
         ↪ {1,3}\.\d{1,3})/)
94      {
95      debug("test␣$guestconf{$prguest}{$prnic}␣!=␣$2\n")
          ↪ ;
96      if ($guestconf{$prguest}{$prnic} ne $2)
97       {
98       $guestconf{$prguest}{'compliant'} = 'not␣
           ↪ compliant';
99       }#if
100     #print " nic = $1 ip = $2";
101      }#if ($vcmdline =~ /(eth\d)\s*(\d{1,3}\.\d
           ↪ {1,3}\.\d{1,3}\.\d{1,3})/)
102     }# while (my $vcmdline = <VCMD>)
103     close(VCMD);
104
105   }#if ($prattribut =~ /(eth\d)/)
106   #Running test for desierd prosesses.
107   elsif($prattribut =~ /(ps\d)/)
108    {
109    my $prps = $1;
110    debug("Prosessing␣prosess␣=␣$prps␣\n");
```

```
111     debug("Running␣command␣python␣vol.py␣-l␣vmi://
            ↪ $prguest␣--profile=$guestconf{$prguest}{'
            ↪ profile'}␣linux_pslist␣|\n");
112     print "INFO␣:␣Framework␣used␣to␣run␣test␣=␣";
113     my $pscompliant = "not␣compliant";
114     open (VCMD, "python␣vol.py␣-l␣vmi://$prguest␣--
            ↪ profile=$guestconf{$prguest}{'profile'}␣
            ↪ linux_pslist␣|");
115     while (my $vcmdline = <VCMD>)
116      {
117      if ($vcmdline =~ /0x[0-9a-f]*\s(\S*)/)
118       {
119       my $prtestvalue = $1;
120       if ($guestconf{$prguest}{$prps} eq $prtestvalue)
121        {
122        $pscompliant = 'compliant';
123        }# if ($guestconf{$prguest}{$prps} eq
               ↪ $prtestvalue)
124       }#if ($vcmdline =~ /0x[0-9a-f]*\s(\S*)/)
125     } #while (my $vcmdline = <VCMD>)
126     close(VCMD);
127     if ($pscompliant eq 'not␣compliant') #if the
            ↪ prosess was not running the guest is not
            ↪ copliant.
128      {
129      $guestconf{$prguest}{'compliant'} = 'not␣
             ↪ compliant';
130      }#if ($pscompliant eq "not compliant";)
131      } # elsif($prattribut =~ /(ps\d)/)
132   #Running test for desierd kernel
133   elsif($prattribut =~ /kernel/)
134    {
135    debug("Prosessing␣$prattribut␣=␣$guestconf{$prguest
           ↪ }{'kernel'}␣\n");
136    debug("python␣vol.py␣-l␣vmi://$prguest␣--profile=
           ↪ $guestconf{$prguest}{'profile'}␣linux_banner"
           ↪ );
137    open (VCMD, "python␣vol.py␣-l␣vmi://$prguest␣--
           ↪ profile=$guestconf{$prguest}{'profile'}␣
           ↪ linux_banner␣|");
138    while (my $vcmdline = <VCMD>)
139     {
140     if ($vcmdline =~ /Linux\sversion\s(\d{1,2}\.\d
           ↪ {1,2}\.\d{1,2}-\d{1,2}-\S*)/)
141      {
142      my $pskernelvr = $1;
143      debug("Run␣test␣$guestconf{$prguest}{'kernel'}␣ne
           ↪ ␣$pskernelvr␣\n");
```

```
144       if ($guestconf{$prguest}{'kernel'} ne $pskernelvr
              ↪ )
145        {
146        $guestconf{$prguest}{'compliant'} = 'not␣
              ↪ compliant';
147        }#if ($guestconf{$prguest}{'kernel'} ne
              ↪ $pskernelvr)
148       }#if
149      }#while (my $vcmdline = <VCMD>)
150     close(VCMD);
151     }#elsif($prattribut =~ /(kernel)/)
152    }#foreatch
153    debug ("test␣$guestconf{$prguest}{'compliant'}␣=␣1\n
          ↪ ");
154    if ($guestconf{$prguest}{'compliant'} eq 'compliant'
          ↪ )
155     {
156     print 'Guest␣',$prguest,'␣is␣',colored ['green␣
          ↪ on_black'],'␣compliant',"\n";
157     }
158    else
159     {
160     print 'Guest␣',$prguest,'␣is␣',colored ['red␣
          ↪ on_black'],'␣not␣compliant',"\n";
161    }
162 }#foreach my $prguest (keys %guestconf)
163 my $elapsed =  Time::HiRes::tv_interval($start);
164 print LOG "$elapsed␣\n";
165
166
167 # print "Tee script executed in $elapsed microseconds
        ↪ \n";
168
169 if ( $opt{'l'} ){
170         close LOG;
171 }
172 close CONF;
173
174 debug("Scripting␣is␣finished,␣exiting...\n");
175
176 exit 0;
177
178
179
180 ######################
181 # subroutines        #
182 ######################
183
```

```
184 sub usage {
185     print "Usage:\n";
186     print "-h for help\n";
187     print "-v for verbose ( more output )\n";
188     print "-d for debug ( even more output )\n";
189     print "-c <filename> for the configuration file\n"
        ↪ ;
190     print "-l <filename> for the log file\n";
191 }
192
193 sub verbose {
194     print "VERBOSE: " . $_[0] if ( $VERBOSE or $DEBUG
        ↪ );
195
196 }
197
198 sub debug {
199     print "DEBUG: " . $_[0] if ( $DEBUG );
200
201 }
```

Appendix C

Resource consuming process Script. rescons.pl

```
1
2
3
4  #!/usr/bin/perl
5
6  # our needed packages
7  use strict "vars";
8  use Getopt::Std;
9  use Time::HiRes;
10
11 # Global variables
12 my $VERBOSE = 0;
13 my $DEBUG = 0;
14 my $NLOOP = 1;
15
16 # sommandline options
17
18 my $opt_string = "vdhn:l:z:"; # print out help message
19 getopts( "$opt_string", \my %opt ) or usage() and exit
       ↪ (1); #or die "Read the manual\n";
20
21 $VERBOSE = 1 if $opt{'v'};
22 $DEBUG = 1 if $opt{'d'};
23 if ( $opt{'h'} ){
24      usage();
25      exit 0;
26 }
27
28 if ($opt{'n'}){
29   $NLOOP = $opt{'n'};
30 }
31
```

```
32 my $LFILE = $opt{'l'};
33 debug("LogFile was $LFILE\n");
34 my $ZFILE = $opt{'z'};
35 debug("File to zip  was $ZFILE\n");
36
37 ( usage() and die "Please supply a file to zip \n" )
     ↪ unless stat($ZFILE);
38
39
40 ##########################
41 #                        #
42 #                        #
43 #       Main part        #
44 #                        #
45 #                        #
46 ##########################
47
48 my $start = [Time::HiRes::gettimeofday()];
49 open(LOG,">$LFILE");
50 my $count =0;
51
52 while ($count++ < $NLOOP) {
53         my $zipstart = [Time::HiRes::gettimeofday()];
54         system ("zip $ZFILE.zip $ZFILE");
55         my $zipelapsed =  Time::HiRes::tv_interval(
             ↪ $zipstart);
56         print LOG "ZIPELAPST = $zipelapsed \n";
57         system ("rm $ZFILE.zip");
58 }#while
59
60
61
62 my $elapsed =  Time::HiRes::tv_interval($start);
63 print LOG "TOTALELAPST = $elapsed \n";
64 close(LOG);
65 debug("Scripting is finished, exiting...\n");
66 exit 0;
67
68
69
70 ######################
71 # subrutines         #
72 ######################
73
74 sub usage {
75     print "Usage:\n";
76     print "-h for help\n";
77     print "-v for verbose ( more output )\n";
```

```
78      print "-d for debug ( even more output )\n";
79      print "-l <filename> for the log file\n";
80      print "-z <filename> to fil to zip\n";
81      print "-n number of loops \n";
82  }
83
84  sub verbose {
85      print "VERBOSE: " . $_[0] if ( $VERBOSE or $DEBUG
            ↪ );
86
87  }
88
89  sub debug {
90      print "DEBUG: " . $_[0] if ( $DEBUG );
91
92  }
```

Appendix D

Scheduling Script. schedule.pl

```
1  #!/usr/bin/perl
2
3  # our needed packages
4  use strict "vars";
5  use Getopt::Std;
6  use Time::HiRes;
7
8  # Global variables
9  my $VERBOSE = 0;
10 my $DEBUG = 0;
11 my $NLOOP = 1;
12 my $FLOOP = 1;
13 # sommandline options
14
15 my $opt_string = "vdhn:f:l:"; # print out help message
16 getopts( "$opt_string", \my %opt ) or usage() and exit
       ↪ (1); #or die "Read the manual\n";
17
18 $VERBOSE = 1 if $opt{'v'};
19 $DEBUG = 1 if $opt{'d'};
20
21 if ( $opt{'h'} ){
22     usage();
23     exit 0;
24 }
25
26 if ($opt{'n'}){
27   $NLOOP = $opt{'n'};
28 }
29
30 if ($opt{'f'}){
31   $FLOOP = $opt{'f'};
32 }
33
```

```
34 my $LFILE = $opt{'l'};
35 debug("LogFile was $LFILE\n");
36
37 ##########################
38 #                        #
39 #                        #
40 #       Main part        #
41 #                        #
42 #                        #
43 ##########################
44
45
46 my $start = [Time::HiRes::gettimeofday()];
47 open(LOG,">$LFILE");
48 my $count =0;
49 my $fcount =0;
50 my $tsleep = 60 / $FLOOP;
51 while ($count++ < $NLOOP) {
52         while ($fcount++ < $FLOOP ){
53                 my $ptstart = time;
54                 my $pstart = [Time::HiRes::
                       ↪ gettimeofday()];
55                 system ("perl compliant.pl -c des.cfg"
                       ↪ );
56                 my $pelapsed =  Time::HiRes::
                       ↪ tv_interval($pstart);
57                 print LOG "PROSESSELAPST = $pelapsed \
                       ↪ n";
58                 my $ptelapsed = time - $ptstart;
59                 sleep ($tsleep - $ptelapsed);
60                 #print "$ptelapsed \n";
61                 $count++;
62         }#while
63         $fcount=0;
64         $count--;
65 }#while
66
67
68
69 my $elapsed =  Time::HiRes::tv_interval($start);
70 print LOG "TOTALELAPST = $elapsed \n";
71 close(LOG);
72 debug("Scripting is finished, exiting...\n");
73 exit 0;
74
75
76
77 ######################
```

```
78 # subrutines          #
79 ######################
80
81 sub usage {
82     print "Usage:\n";
83     print "-h for help\n";
84     print "-v for verbose ( more output )\n";
85     print "-d for debug ( even more output )\n";
86     print "-l <filename> for the log file\n";
87     print "-n number of loops\n";
88     print "-f number of loops/min \n";
89 }
90
91 sub verbose {
92     print "VERBOSE: " . $_[0] if ( $VERBOSE or $DEBUG
       ↪ );
93
94 }
95
96 sub debug {
97     print "DEBUG: " . $_[0] if ( $DEBUG );
98
99 }
```

Appendix E

Data from the experiments

E.1 Data Experiment 1

Example network configuration file. /etc/network/interfaces

```
iface eth0 inet static
        address 172.24.201.85
        netmask 255.255.255.0
        gateway 172.24.201.1
        dns-nameservers 172.24.201.51 172.24.201.50
```

E.2 Data Experiment 2

47.761761	48.673629	48.436537	47.449149	48.676068
47.06424	48.393583	48.510077	47.675399	49.277353
47.637454	47.233528	49.292919	47.452949	49.379888
48.758873	47.334911	48.281828	48.302791	47.966527
49.03947	47.406834	48.35909	49.119929	47.244834
49.226746	47.539088	48.642393	49.078148	47.559358
47.08533	48.93096	47.602828	48.884824	47.190709
49.79394	47.839685	48.247879	47.579723	47.284758
48.41433	47.84694	48.76632	49.366785	48.796458
47.218244	48.21756	48.296392	47.739699	47.092072

Table E.1: Experiment 2: Baseline Control Sample

47.454471	47.937916	48.066164	48.151241	49.075062
47.491533	47.59132	48.292201	47.91982	48.851663
46.771776	48.297171	47.549821	47.820402	47.423816
48.743196	47.286723	47.986503	49.215293	47.28157
48.299666	47.503254	47.872113	48.674127	47.108775
48.987687	47.10709	47.41061	47.685822	48.333484
47.313125	47.529352	48.969366	47.571536	49.040809
47.35318	48.245409	48.954029	46.990614	48.31121
47.513019	47.995368	48.631588	47.53765	48.307207
47.639704	48.260211	48.0943	47.974323	48.71104

Table E.2: Experiment 2: 1 test / min

47.50316	47.738229	49.819227	47.258294	48.523667
46.571925	47.601103	46.933805	47.47359	47.303402
47.927472	46.827176	46.875655	47.454315	46.472014
48.207588	46.836022	47.584819	47.507105	47.31375
47.710461	47.47127	47.7165	47.49331	47.499902
47.341948	47.123104	47.223047	48.683462	46.503461
47.239606	47.215106	47.823663	47.355603	47.565556
47.645448	48.314218	46.712412	48.361928	47.037207
47.457129	48.428765	47.112992	47.073549	47.201932
47.562757	47.170382	47.802442	46.968339	47.329637

Table E.3: Experiment 2: 2 test / min

46.891185	48.928168	46.332371	47.905114	47.301139
48.036501	47.542456	48.400814	46.273667	48.254983
46.765992	47.759924	47.523362	48.679044	47.361048
47.983696	46.885332	47.863652	47.616058	48.195861
47.143221	47.489797	46.761929	47.699706	47.061547
47.903794	46.701512	46.983753	47.810413	46.765693
48.258159	47.324259	47.439066	47.411794	47.612629
48.015506	47.074336	47.374155	47.15397	47.609163
46.685609	48.037735	47.079752	47.821692	46.756021
47.318515	47.641179	47.845502	46.593105	47.420625

Table E.4: Experiment 2: 3 test / min

46.562385	47.852243	46.611621	46.883082	47.091203
46.391071	46.768588	47.676529	46.541507	46.693917
46.705503	46.60567	47.001847	46.981602	46.370436
47.047143	47.073911	46.486825	46.422632	46.934454
47.150423	46.93365	47.135628	47.38684	47.226514
46.474936	46.71525	46.977861	47.083291	47.485774
46.898286	48.179528	46.815653	48.358582	46.446543
48.541468	46.861018	49.014328	46.40236	48.694171
46.577788	47.442013	47.506804	46.537776	47.662722
47.110569	47.47577	46.719857	47.285444	47.033288

Table E.5: Experiment 2: 4 test / min

E.3 Data Experiment 3

4.797579	4.809241	4.828273	4.836781	4.840397
4.83321	4.814599	4.869325	4.790027	4.830524
4.878967	4.831715	4.823304	4.905062	4.831158
4.851802	4.851571	4.832396	4.822973	4.82912
4.848443	4.838408	4.837767	4.859769	4.860369
4.852312	4.853774	4.853134	4.844499	4.887942
4.848591	4.829867	5.270561	5.289274	4.826391
4.839661	5.406341	5.13644	4.83129	4.876589
4.857804	4.848966	4.817508	4.90517	4.829175
4.832373	4.811563	4.837398	4.854046	5.54132

Table E.6: Experiment 3: Performance Volatility Process List.

0.131922	0.133965	0.125096	0.132217	0.132877
0.129779	0.133419	0.131282	0.129941	0.129595
0.131827	0.130093	0.138299	0.129648	0.131734
0.136421	0.127076	0.130902	0.128363	0.128507
0.129099	0.129458	0.129291	0.131467	0.128844
0.12994	0.129396	0.131464	0.129715	0.131552
0.13138	0.128991	0.127931	0.130868	0.13203
0.129716	0.13535	0.128142	0.131405	0.131497
0.128576	0.132028	0.132117	0.130025	0.130287
0.130453	0.130609	0.130124	0.128576	

Table E.7: Experiment 3: Performance libvmi Prosess list.

E.4 Performance Resource consuming process.

49.846953	52.182269	48.967622	49.487107	50.430052
50.672137	51.616574	50.662909	48.952976	49.856644
48.804827	50.727033	50.233826	50.726492	50.098972
49.536628	50.348717	50.235335	49.87225	51.624058
49.840646	50.691398	48.677032	48.18498	48.797462
48.590696	48.869346	50.017258	49.850488	48.9247
50.420325	51.036269	50.227535	48.129617	50.823237
49.651466	50.672091	49.747063	50.411376	51.013217
50.553519	51.144716	50.017074	49.141522	51.094471
48.545863	49.772129	47.977078	51.409851	52.701591

Table E.8: Experiment 4: Performance Resource consuming process at 60 interrupt a minute.

57.463015	58.852671	59.476685	57.572822	59.737042
59.573163	60.080721	59.781118	59.144315	60.556481
59.836314	59.002631	58.023553	59.766966	60.50039
60.029929	59.729923	58.947662	59.825212	59.829398
57.974503	59.003986	57.916768	59.664481	59.263697
60.111073	58.639836	59.276279	60.740703	59.06033
59.375331	58.435146	57.477703	58.787827	58.093023
60.572501	56.977618	58.508955	59.55781	60.037519
59.028995	58.672216	59.608968	59.950713	58.724172
58.716035	59.024913	58.030031	58.838024	59.072785

Table E.9: Experiment 4: Performance Resource consuming process at 120 interrupt a minute.

72.734258	70.253542	68.17171	65.136669	64.631862
62.943288	66.060292	65.635658	66.546737	72.046329
71.731202	66.396613	63.714913	64.719045	64.137432
63.97184	65.228439	67.771699	71.522405	71.35151
65.159445	64.598772	66.055241	64.427392	65.710647
64.563254	68.221273	70.573493	73.37364	63.738532
62.921897	65.163702	64.643233	64.687701	64.775357
68.599715	74.035134	70.417433	65.271492	64.370269
65.583827	65.955791	64.321054	64.120511	72.879448
72.505999	67.295417	65.468217	65.263887	64.915537

Table E.10: Experiment 4: Performance Resource consuming process at 180 interrupt a minute.

88.73857	76.89636	70.202496	82.569784	83.125389
71.683805	78.4402	86.385807	70.943457	72.835241
88.468576	75.357286	70.254922	82.113924	82.471432
71.433961	76.779996	88.086622	70.432444	70.811157
86.419854	78.628507	71.594937	78.486308	85.33365
70.263938	72.824242	85.413028	76.904734	69.770777
81.358191	84.442342	70.447295	72.739251	89.729221
73.448772	70.831516	82.771949	81.256602	71.531394
77.05528	87.598071	71.592883	76.57183	88.60263
72.273971	71.510966	86.734697	78.468791	70.931852

Table E.11: Experiment 4: Performance Resource consuming process at 240 interrupt a minute.

103.219395	83.656075	101.504258	82.54433	99.179459
80.691867	97.505189	85.042491	98.170028	86.644256
93.175022	88.015218	92.630428	88.719199	91.150997
89.773232	92.238778	90.740126	90.687814	98.441581
84.337993	98.910622	83.291694	102.718798	82.179905
99.749892	85.581016	97.536277	87.887231	94.269737
91.459928	90.976704	91.147361	89.456094	94.239937
87.342391	96.071854	85.862699	103.043992	84.007375
102.418969	87.492086	94.527782	89.90048	93.187904
96.291633	87.738028	97.850085	81.990925	97.539459

Table E.12: Experiment 4: Performance Resource consuming process at 300 interrupt a minute.

113.797553	109.775641	111.601199	111.547135	109.865061
113.440477	108.116212	116.117576	114.262647	111.486272
111.594907	113.28485	114.283764	114.046268	115.371431
113.454608	113.27793	113.460296	116.040363	116.108266
108.243761	113.71759	112.972115	113.915695	113.75621
112.611201	116.371403	110.864881	111.049944	111.592666
109.815897	109.384353	113.933158	114.198314	111.72956
116.085729	108.858078	109.480623	112.359313	114.90916
113.078547	113.84938	115.139603	110.415952	114.096058
110.86694	112.84468	112.197091	110.214499	112.547694

Table E.13: Experiment 4: Performance Resource consuming process at 360 interrupt a minute.

Bibliography

[1] Benjamin Armstrong. *VMMs versus Hypervisors*. July 2006. URL: http://blogs.msdn.com/b/virtual_pc_guy/archive/2006/07/10/661958.aspx.

[2] Brendan Dolan-Gavitt, Bryan Payne and Wenke Lee. *Leveraging Forensic Tools for Virtual Machine Introspection*. 2011.

[3] Brendan Dolan-Gavitt et al. 'Virtuoso: Narrowing the Semantic Gap in Virtual Machine Introspection'. In: *Security and Privacy (SP), 2011 IEEE Symposium on* (May 2011), pp. 297–312.

[4] Lawrence D'Oliveiro et al. *Protection ring*. URL: http://en.wikipedia.org/wiki/Protection_ring.

[5] *Driver Domain*. URL: http://wiki.xen.org/wiki/Driver_Domain.

[6] George Dunlap. *An Introduction to Full Virtualization With Xen*. Oct. 2012. URL: http://www.linux.com/news/enterprise/systems-management/655446-an-introduction-to-paravirtualization-and-xen.

[7] Yangchun Fu and Zhiqiang Lin. 'EXTERIOR: Using Dual-VM Based External Shell for Guest-OS Introspection, Configuration, and Recovery'. In: *VEE '13 Proceedings of the 9th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments* (2013), pp. 97–110.

[8] Yangchun Fu and Zhiqiang Lin. 'Space Traveling across VM: Automatically Bridging the Semantic Gap in Virtual Machine Introspection via Online Kernel Data Redirection'. In: *Security and Privacy (SP), 2012 IEEE Symposium on* (May 2012), pp. 586–600. ISSN: 1081-6011.

[9] *Hardware-assisted virtualization*. URL: http://en.wikipedia.org/wiki/Hardware-assisted_virtualization.

[10] S. T. Jones, A. C. Arpaci-Dusseau and R. H. Arpaci-Dusseau. *Antfarm: tracking processes in a virtual machine environment*. In Proceedings of the annual conference on USENIX 06 Annual Technical Conference, Mar. 2006.

[11] S. T. Jones, A. C. Arpaci-Dusseau and R. H. Arpaci-Dusseau. 'VMM-based Hidden Process Detection and Identification using Lycosid'. In: *Proceeding VEE 08 Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments* (2008), pp. 91–100.

[12] Ashlesha Joshi et al. 'Detecting Past and Present Intrusions through Vulnerability-Specific Predicates'. Version Volume 39 Issue 5. In: *SOSP 05 Proceedings of the twentieth ACM symposium on Operating systems principles* (Dec. 2005). Volume 39 Issue 5, pp. 91–104.

[13] Tamas K. Lengyel et al. *Virtual Machine Introspection in a Hybrid Honeypot Architecture*. 2012. URL: https://www.usenix.org/conference/cset12/workshop-program/presentation/Lengyel.

[14] 'LibVMIIntroduction'. In: (). URL: https://code.google.com/p/vmitools/wiki/LibVMIIntroduction.

[15] Zhiqiang Lin, Xiangyu Zhang and Dongyan Xu. *Automatic Reverse Engineering of Data Structures from Binary Execution*. Feb. 2010. URL: https://www.utdallas.edu/~zxl111930/file/Rewards_NDSS10.pdf.

[16] Lionel Litty and David Lie. 'Manitou: a layer-below approach to fighting malware'. In: *ASID 06 Proceedings of the 1st workshop on Architectural and system support for improving software dependability* (2006), pp. 6–11.

[17] name1, name2 et al. *System.map*. URL: http://en.wikipedia.org/wiki/System.map.

[18] K. Nance, M. Bishop and B. Hay. *Virtual Machine Introspection: Observation or Interference*. Sept. 2008, pp. 32–37.

[19] Jason Nash. *vSphere Security: A Tour of the vSphere vShield Suite*. URL: http://blog.pluralsight.com/vsphere-vshield.

[20] 'Official Archive Mirrors for Ubuntu'. In: (). URL: https://launchpad.net/ubuntu/+archivemirrors.

[21] Oracle. *VirtualBox*. URL: https://www.virtualbox.org/.

[22] White Paper. 'Understanding Full Virtualization, Paravirtualization, and Hardware Assist'. In: (2007). URL: http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf.

[23] B. D. Payne, M. Carbone and W. Lee. *Secure and flexible monitoring of virtual machines*. In Proceedings of the 23rd Annual Computer Security Applications Conference, Dec. 2007.

[24] Bryan D. Payne. *PyVMI*. URL: https://github.com/libvmi/libvmi/tree/master/tools/pyvmi.

[25] Bryan D. Payne. 'Simplifying Virtual Machine Introspection Using LibVMI'. In: *Sandia Report* (2012).

[26] Danielle Ruest. *Virtualization hypervisor comparison: Type 1 vs. Type 2 hypervisors*. URL: http://searchservervirtualization.techtarget.com/tip/Virtualization-hypervisor-comparison-Type-1-vs-Type-2-hypervisors.

[27] *Stuxnet*. URL: http://en.wikipedia.org/wiki/Stuxnet.

[28] Sahil Suneja et al. 'Non-intrusive, Out-of-band and Out-of-the-box Systems Monitoring in the Cloud'. In: *ACM SIGMETRICS Performance Evaluation Review - Performance evaluation review* (June 2014). Volume 42 Issue 1, pp. 249–261.

[29] *Trend Micro Deep Security 7.5*. URL: https://ca-uat1.insight.com/content/dam/insight/en_US/pdfs/trend-micro/Deep-Security-75-Datasheet.pdf.

[30] vmware. *Virtualization*. URL: http://www.vmware.com/virtualization.

[31] vmware. *VMware Workstation*. URL: https://www.vmware.com/products/workstation/.

[32] *VMware vShield. Virtualization-Aware Security for the Cloud*. URL: http://www.vmware.com/files/pdf/vmware-vshield_br-en.pdf.

[33] *Volatility Foundation*. URL: http://www.volatilityfoundation.org.

[34] *Xen Project Best Practices*. URL: http://wiki.xenproject.org/wiki/Xen_Project_Best_Practices.

[35] *Xen Project Software Overview*. URL: http://wiki.xen.org/wiki/Xen_Overview#PV_on_HVM.

[36] Haiquan Xiong et al. 'Libvmi: A Library for Bridging the Semantic Gap between Guest OS and VMM'. In: (2012), pp. 549–556.