NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

PROTECTING COMPROMISED SYSTEMS WITH A VIRTUAL-MACHINE PROTECTION AND CHECKING SYSTEM USING OUT-OF-GUEST PERMISSIONS

by

Alexis Peppas

December 2017

Thesis Advisor: Geoffry Xie
Second Reader: Charles Prince

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704–0188

1. AGENCY USE ONLY *(Leave Blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 2017	Master's Thesis MM-DD-YYYY to MM-DD-YYYY

4. TITLE AND SUBTITLE

PROTECTING COMPROMISED SYSTEMS WITH A VIRTUAL-MACHINE PROTECTION AND CHECKING SYSTEM USING OUT-OF-GUEST PERMISSIONS

5. FUNDING NUMBERS

6. AUTHOR(S)

Alexis Peppas

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Postgraduate School
Monterey, CA 93943

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

N/A

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

The views expressed in this document are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol Number: N/A.

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release. Distribution is unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

14. SUBJECT TERMS

15. NUMBER OF PAGES 51

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2–89)
Prescribed by ANSI Std. 239–18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited

PROTECTING COMPROMISED SYSTEMS WITH A VIRTUAL-MACHINE PROTECTION AND CHECKING SYSTEM USING OUT-OF-GUEST PERMISSIONS

Alexis Peppas
Lt, Navy
B.S., Hellenic Naval Academy, 2003

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 2017

Approved by: Geoffry Xie
Thesis Advisor

Charles Prince
Second Reader

Peter Denning
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

THIS PAGE INTENTIONALLY LEFT BLANK

List of Figures

THIS PAGE INTENTIONALLY LEFT BLANK

List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

List of Acronyms and Abbreviations

NIST	National Institute of Standards and Technology
OS	Operating System
VM	Virtual Machines
CPU	Central Processing Unit
VMI	Virtual Machine Introspection
ACL	Access Control List
SACL	Shadow ACL
VMPCS-OGP	Virtual-Machine Protection and Checking System Using Out-Of-Guest Permissions
API	Application program interface
VT	Virtualization Technology
PT	Page Table
EPT	Extended Page Tables
GMFN	Guest Machine Frame Number
MFN	Machine Frame Number
IOMMU	Input/Output Memory Management Unit
GVA	Guest Virtual Address
MAC	Mandatory Access Control
HAP	High Assurance Processes
OI	Object Identifiers

SGX	Software Guard Extensions
IDS	Intrusion Detection System
HIDS	Host Intrusion Detection System (IDS)
NIDS	Network IDS
NIC	Network Interface Card

Executive Summary

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

THIS PAGE INTENTIONALLY LEFT BLANK

Acknowledgments

I would like to thank ...

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:
Introduction

System virtualization has been increasing in popularity over the last years. It makes it possible to run many and different Operating Systems (OS) on the same physical machine. That kind of Operating System (OS), known as Virtual Machines (VM), is run independently of each other on the same physical machine, known as host, without any indication that there is another OS running on the same Host. It is essentially a resource sharing mechanism. The software that facilitates this capability is called a hypervisor.

The emergence of cloud computing has increased the requirement of many new and different services from different vendors, usually around the globe. According to the National Institute of Standards and Technology (NIST) [1], "cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction".

Virtualization gave a solution to the increasing requirement of resources for these services to run on. Instead of having many separate physical machines running the required different software, usually resulting in underutilization, one machine, with better specifications, was used, and with the use of virtualization, each vendor could run his services on a dedicated VM.

Also, in order to improve network security on these machines, as well as redundancy among different services, instead of having one VM running all the services required by one vendor, service providers started using many different VMs, each requiring less resources and running one or just a couple of services. This way, if one VM fails, the rest of the services keep running. Furthermore, by having each VM run only a few services, the attack surface available for possible vulnerability exploitation is reduced significantly.

This increase in the use of virtualization has driven even hardware manufacturers, like Intel and AMD, to introduce special virtualization Central Processing Unit (CPU) instructions,

to facilitate better, more reliable and secure allocation, sharing, usage and performance of VMs.

Despite the evolution of CPU virtualization instructions, and the continuous development of more efficient and secure hypervisors, the bottom-line remains the same. A VM is still a system, with all the vulnerabilities of its running OS and software, which at some point in time, will be the victim of a successful exploitation.

1.1 Problem Statement

The motivation for this research came from that idea exactly. When an OS is running directly on a physical machine, it is on its own to allocate and use its resources and protect itself from network or other types of attacks. But, when it runs on a virtualization platform, the hypervisor stands between the hardware and the running software, and has full visibility in what is happening inside a VM.

The native Linux file permission system, although simple to manage and efficient, it lacks fine-grained user/group access to files. Once a user belongs to a group, nothing prohibits him from accessing all the files accessible to that group. Furthermore, when an attacker gains access to a system, he usually will try to escalate his privileges by having access to the root account. From that point there is nothing out of reach and the attacker has unrestricted access to the entire system and is free to read and modify files and change the system's configuration to his liking, to serve his purposes.

In [2], the author names a new technique which leverages this viewing ability of the hypervisor. Virtual Machine Introspection (VMI) is the "approach of inspecting a virtual machine from the outside for the purpose of analyzing the software running inside it". Having in mind that a system will be eventually subverted, we want to leverage the introspection capability of a hypervisor to try to protect critical files for the OS, the user or both. We want to create an gut-of-guest Access Control List (ACL), which we call Shadow ACL (SACL), for managing file access inside a VM. We call this mechanism Protecting Compromised Systems with a Virtual-Machine Protection and Checking System Using Out-Of-Guest Permissions (VMPCS-OGP).

In our research, we develop a prototype for file access monitor and control outside a VM.

We use a 64-bit Ubuntu OS running on top of a Xen hypervisor. The prototype leverages the VMI capability of the Xen hypervisor leveraged with the LibVMI Application program interface (API) [3], as well as DRAKVUF [4], a system used for dynamic malware analysis. It includes a modified DRAKVUF implementation, as well as prototypes of the ACL, kept on the hypervisor, to be enforced on the guest VM. Our approach is to provide a more tightened but fine-grained environment.

In this work we will try to assess how we can leverage the introspection capabilities of the Xen hypervisor to improve the OS built-in confidentiality and integrity mechanisms. Some of these cases include denying access to the root user, who has access to the entire filesystem. We want to make a more fine-grained access control to fill the gap of the Linux native permission bits, by denying access to files on users that belong to a group with access. Furthermore, we want to alter the user permissions, by keeping a SACL.

1.2 Research Questions

The primary issue we will address in this research is whether we can enforce out-of-guest permissions to check access to the files of a system, so that the attacker is not able to read or write critical files on the system. Following that we will address:

- What is the best way to implement a monitor for file access on the guest.
- What is the performance overhead.
- If this mechanism can be leveraged to identify a compromised system or a system actively being compromised.
- If it is manageable to monitor all files on a system or only specific ones.
- What is the best way to implement VMPCS-OGP on a guest and still provide usability and protection.
- If VMPCS-OGP can be used to discover how a system was compromised and attacker methods in compromising a system – sort of honey pot approach.
- If we can return a valid error to the VM while denying access to a file, so that it does not reveal the extra security check imposed by the hypervisor.

1.3 Organization

This paper is organized into five chapters. Chapter 1 introduces the concepts and thesis focus. Chapter 2 covers some background information for this thesis research used platform, as well as some of the security solution already presented, that make use of VMI. Chapter 3 analyzes the design and methodology of the implemented mechanism, and Chapter 4 discusses the performance testing results and presents our conclusions. Chapter 5 suggests future work.

CHAPTER 2:
Background

In this chapter, the information of the relevant software and hardware is presented. In the first section a short introduction on virtualization and its benefits is given. Then the types of hypervisors and Xen, the platform we will work on, are described. The next section refers to VMI and Xen's capabilities in that field, the LibVMI API and DRAKVUF, the library and main application we will leverage, as well as the system call functionality and convention. Finally, some of the existing solutions that leverage introspection are reviewed.

2.1 Virtualization

Running many and different services on a single OS is not recommended anymore. Cheap hardware lead during the past years in systems that ran a plethora of different software, which became over time a challenge to manage efficiently and securely. Because hardware was inexpensive, service providers, preferred to run a service per physical system to achieve higher security [5]. On the downside, running one service per physical machine, resulted to underutilization of hardware and capabilities, as well as increased cost of maintenance. Virtualization was a solution to the problem [5]. By hosting different VMs on a single and powerful system solves many of the problems. Resources are used efficiently, with each service using only a part of the underlying hardware. Security is implemented easier, as it is much simpler to secure one machine running one service, than having to combine all of them on one. Redundancy between services is also achieved, since each VM is independent from the rest, and any failure does not affect the rest of the VMs.

But the advantages of virtualization do not stop there. Easy backup, restore, cloning and migration of a system are just a few of them. Creating snapshots of entire machines and restoring to a previous state, in case of corruption or misconfiguration, has become a trivial task. Also, modern hypervisors implement a very solid and sophisticated VM isolation, that pivoting from one VM to another, as well as hypervisor attacks, have become extremely difficult.

Hypervisor is the software that drives this mechanism. It runs directly on the hardware

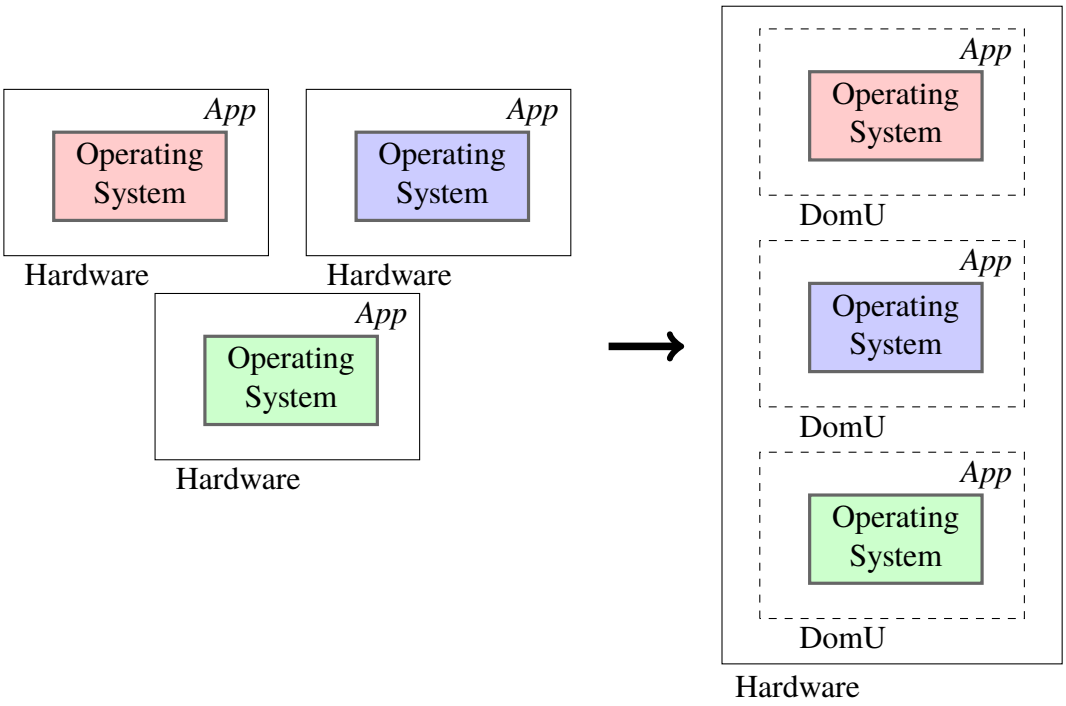


Figure 2.1. Migrating to virtualization

and uses a separate OS installation and resides outside all the guest VMs. At the same time, since the hypervisor manages the allocation and usage of the physical resources, has a unique visibility of the internal state of each VM.

2.1.1 Hypervisor types

Different vendors provide their solution in virtualization. Generally, hypervisors are separated in two categories. Type-I or bare-metal hypervisors and type-II or hosted hypervisors.

Type-II hypervisors are applications, which require a host OS to run on. Typical type-II solution are VMWare Workstation and Oracle VirtualBox. These hypervisors work as any other application and the VMs run on top of them. Although they are simpler to manage for the average user, as well as for simple applications or use as testing environment, type-II hypervisors perform worse than type-I, as explained below.

Type-I hypervisors run directly on the hardware, managing the resources directly without the intervention of any host OS. On the contrary, a more privileged OS is used, to provide an API for the efficient management of the hypervisor and its hosted VMs. Type-I hypervisors

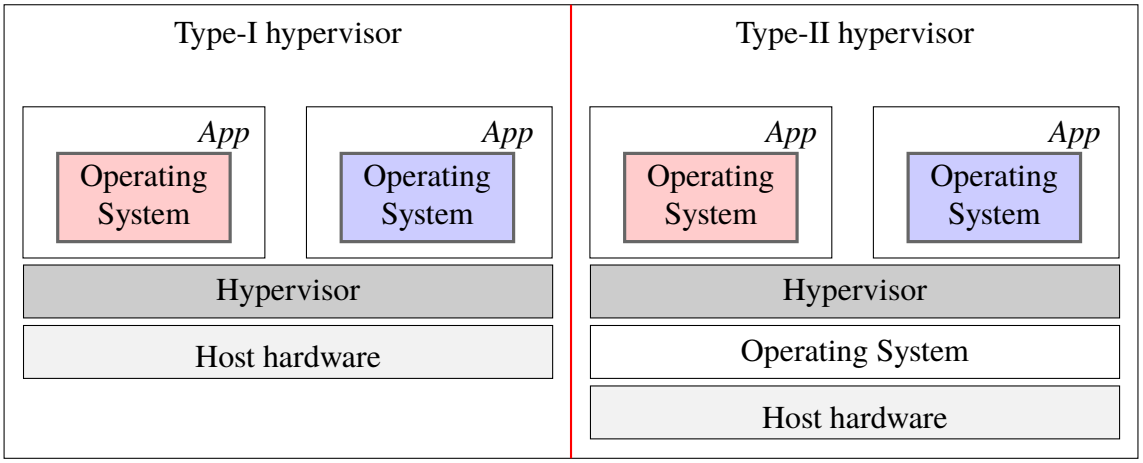


Figure 2.2. Architectural difference of type-I vs type-II hypervisors

are most commonly used in server deployment and enterprise solutions, where performance and efficiency are important. Figure 2.2 shows the basic architectural difference between the two types.

2.1.2 The Xen project

The Xen Project is an open-source type-I hypervisor [6]. Its small footprint and limited interface to the Guest, makes it more robust and secure. The hypervisor runs directly on top of the hardware, as depicted in Figure 2.3. It requires a host OS which acts as an interface between the hypervisor and the user, as well as paravirtualized guests. This host OS is called control or privileged domain, also known as Dom0, and runs at a more privileged level than the rest of the VMs. The rest of the VMs run on a lower privilege level and are called guest domains or DomUs.

To understand how this happens, we need to introduce another CPU architectural feature, which provides different privilege levels for the execution of the CPU instructions, depending on what is the nature of the program invoking them. This mechanism is called protection rings and is present on all modern CPUs and is used from all modern OSs. Protection rings are numbered 0 to 3, with 0 being the most privileged. Usually, applications run in ring 3, also called user mode, and the kernel and device drivers run in ring 3, also called privileged or supervisor mode. But, the hypervisor must run at a more privileged level than the guest OS, in order to allocate and manage the shared resources, otherwise there

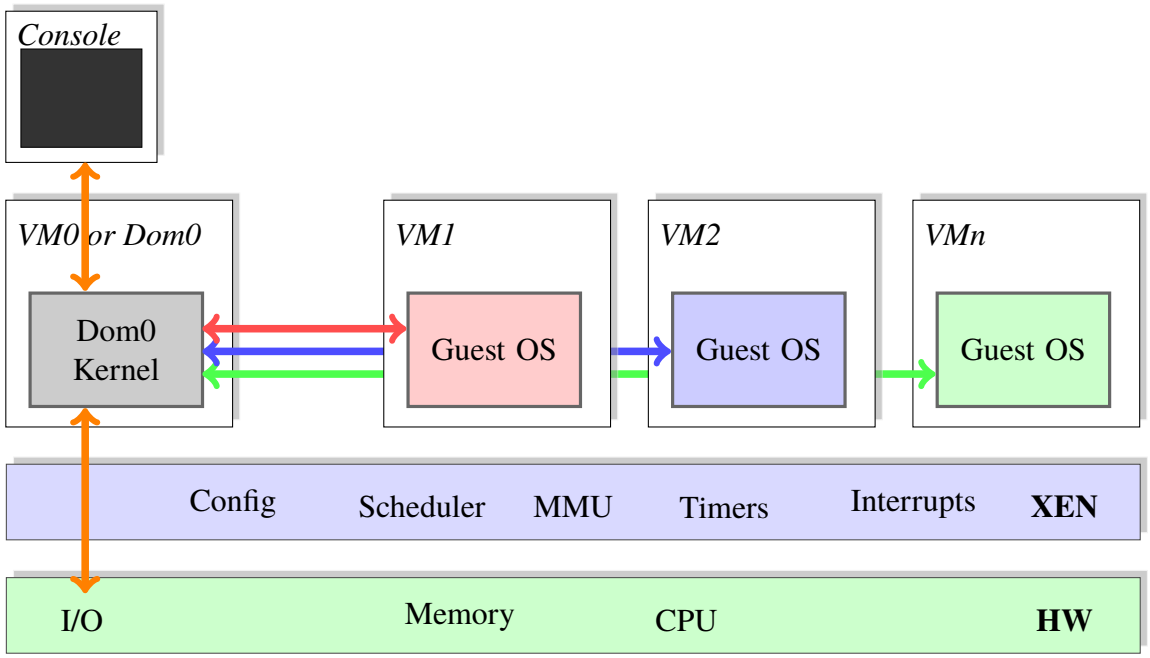


Figure 2.3. Xen Hypervisor Architecture

is a conflict when the guest OS or the hypervisor tries to manage the systems resources. Initially, paravirtualization was used. A technique where OS vendors had to modify their kernels to run on a different privilege level, besides 0, like 1 or 2, to avoid that conflict between the guest OS kernel and the hypervisor.

For type-I hypervisors to work efficiently and without any guest OS modification due to conflicts on the protection ring 0, CPU manufacturers have introduced a new ring -1 to support virtualization. The new ring, called hypervisor mode, is even more privileged than ring 0 and is employed only during hypervisor execution. This new architecture is supported on newer CPUs that employ Virtualization Technology (VT), VT-x for Intel and AMD-V for AMD processors. From the moment CPUs started supporting VT, the employment of VMs started rising significantly.

As virtualization keeps advancing, there is always the question of whether we can leverage it, to provide more than efficient sharing and usage of resources. The unique ability of the hypervisor to access the state of a VM, at a CPU register level or byte of memory, has been the center of research for many years.

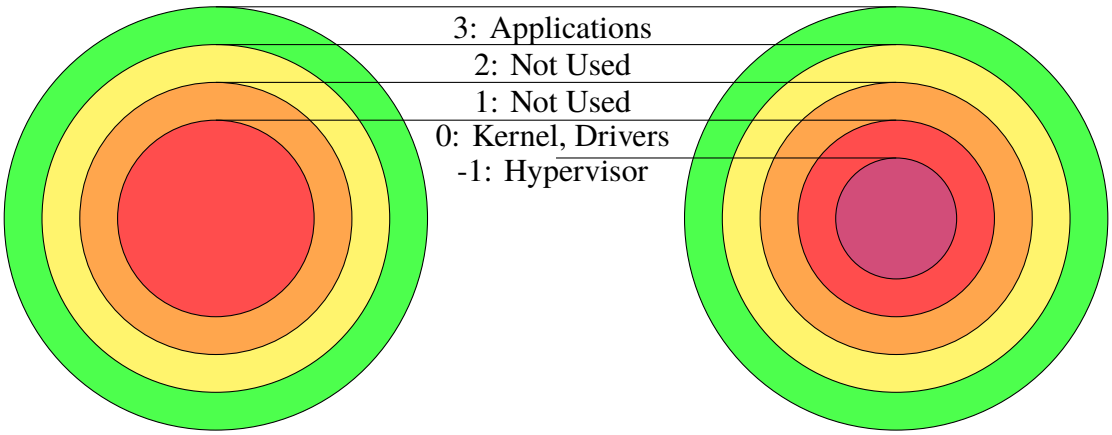


Figure 2.4. x86 protection rings

2.2 Virtual Machine Introspection

As firstly introduced as a concept in [2], VMI is the leverage of the more privileged status of the hypervisor, to inspect the internal state of a VM. The Xen hypervisor, trying to make that more efficient, included introspection methods to inspect its guest VMs. To make these methods more accessible and provide better introspection capabilities, XenAccess [7] was implemented, as well as the mem-events APIs to address that concept [8]. Because of strong research and security interest, introspection in Xen progressed and eventually LibVMI [3] was introduced. It is a library that makes the introspection capabilities of the Xen hypervisor even more accessible. It also provides access to part of the hypervisors introspection methods to third-party applications, using a C or Python interface, the later called PyVMI.

Initially, the memory management of the hypervisor included an extra step in the memory access mechanism. Because each VM assumes that has complete control over the entire address space, and assumes that it writes directly on the hardware, the hypervisor must introduce this extra step. Normally the OS had to translate the virtual address used by an application to a physical address on the hardware. For the hypervisor, each VM is essentially an application. Since every OS will try to write eventually on the same physical address, the hypervisor must make a distinction between the VMs. It assigns each VM a specific physical address space, which then tracks by having additional Page Table (PT) to translate between Guest Machine Frame Number (GMFN) and Machine Frame Number (MFN). This mapping is a one-to-one.

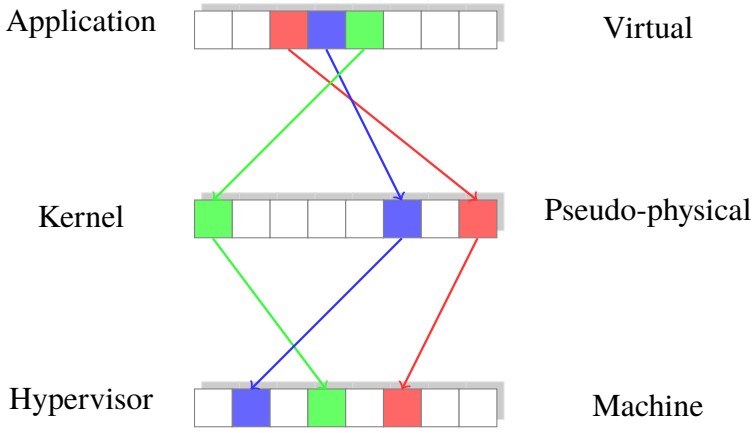


Figure 2.5. Hypervisor memory management concept

With the introduction of Input/Output Memory Management Unit (IOMMU), this extra step got eliminated. Additionally, when Intel, with its Haswell generation CPU, included the support of 512 Extended Page Tables (EPT)s, Xen's introspection capabilities increased, while the overhead reduced significantly. Furthermore, this allowed better isolation and therefore enhanced security between the VMs. Following that development, XenAccess and mem-events were redesigned and were evolved to altp2m, the new Xen VMI subsystem. One of the most critical changes that came with altp2m, was the concurrent assignment of multiple EPTs per VM, a capability which although it was available, was never leveraged. This was a significant improvement, as the hypervisor can keep track of different EPTs with different permissions, which can change during the execution of the VM.

LibVMI, as mentioned earlier, is an API which provides exposure to a subset of Xen's VMI functionalities, as well as other platforms. It makes possible to monitor the state of any VM, including memory and CPU state. Memory can be accessed directly, using physical addresses, or indirectly with the use of virtual addresses, OS and user application symbols. It can monitor memory and register events and provide notifications for them, allowing this way the execution of callback functions.

LibVMI focuses in a subset of introspection methods, that provide memory reading and writing capabilities from running VMs. It provides also methods for accessing and modifying CPU registers, as well as helper methods to pause and unpause a VM. Accessing a VM' memory space is not a trivial task. After detecting where the page directory is, a scan

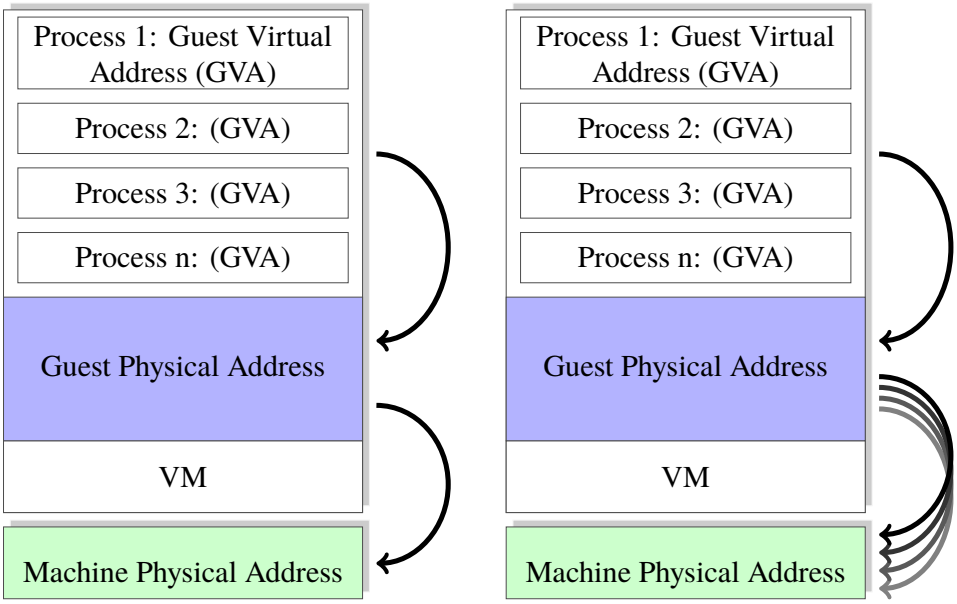


Figure 2.6. Normal vs altp2m multiple EPT assignment

of the page tables follows, to detect the memory mapping of the running process. This gets translated to a virtual address, which later, on the hypervisor gets translated to a physical address. The following figure 2.8 shows a slightly different request, that of reading a kernel symbol.

Xen's introspection methods have a very significant impact on system security. The monitoring application resides on the Host and accesses the VMs state from the hypervisor. That implies a zero-footprint monitoring tool, from the VMs perspective. The monitor does not leave a trace of its action that can be detected from inside the guest.

Although this development was game-changing, it had its drawback. Just monitoring that values of specific parts of memory, or the CPU registers, over a time interval to make any inferences about the running state of the VM, leaves the VM vulnerable during the waiting period. A solution is to trap the memory regions that we want to monitor for access or modification. But this can be detected from a knowledgeable adversary.

To solve this problem the Xen's newest VMI API, altp2m, along with the substantial number of EPTs on the latest CPUs, were employed. This project, DRAKVUF [4], is a dynamic malware analysis platform. One of the most significant key features, is that it traps the

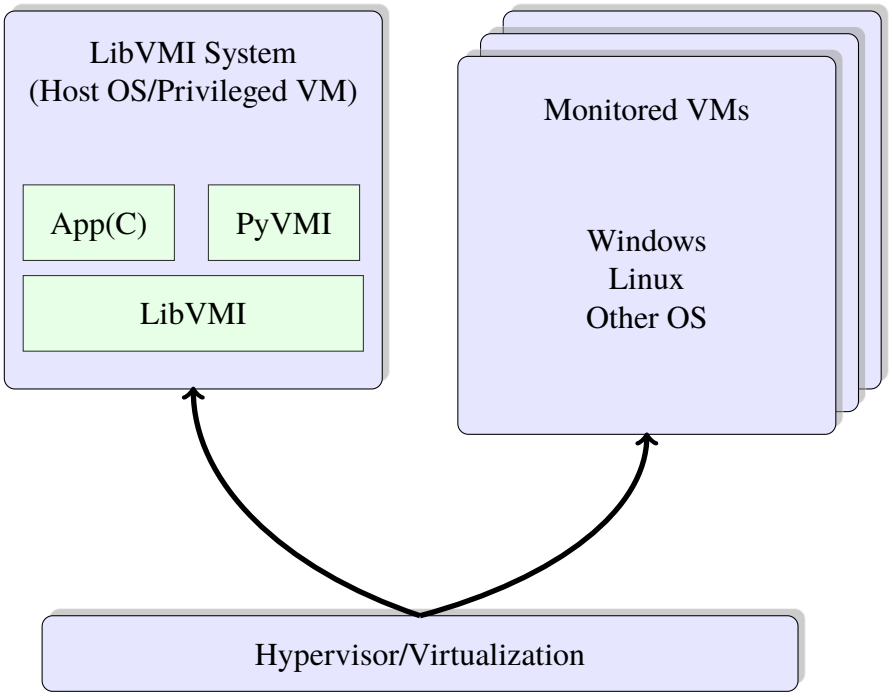
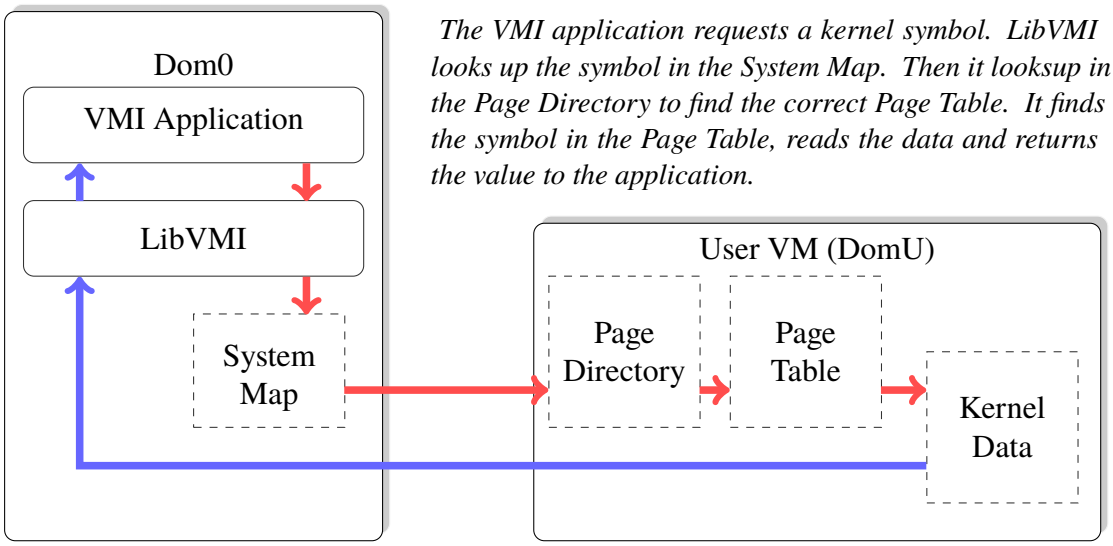


Figure 2.7. LibVMI out of guest access of VM state

memory addresses the user wants to monitor. When the event gets triggered, the EPT with the trapped address gets swapped with the original, continuing that way an unmodified execution of the guest VM. This allows the monitor of an arbitrary number of memory addresses, providing notification on every such event, while at the same time it is untraceable from inside the guest.

A compromised system is just a matter of time. Whether it results from user error, or targeted malicious activity, it is bound to happen. This eventuality led researchers to invest their resources to VM security. Some solutions focus on the analysis part, where by leveraging the hypervisors introspection methods gain better insight and understanding of the behavior and impact of a malware, so that it can be successfully intercepted. Other solutions have a more active role, by trying to protect crucial parts of a running VM. They prevent the kernel from being corrupt, or provide secure access to parts of memory where critical information or applications are stored. These solutions can provide valuable information on which events and actions led to a compromised system, or protect the vital OS space from being corrupt by malicious activity, each of them on its own unique way.



The VMI application requests a kernel symbol. LibVMI looks up the symbol in the System Map. Then it looksup in the Page Directory to find the correct Page Table. It finds the symbol in the Page Table, reads the data and returns the value to the application.

Figure 2.8. Using LibVMI to access the value of a kernel symbol

2.3 System Calls

Modern OSs are responsible for allocating their resources efficiently and securely to themselves, as well as the user level applications. The part of the OS assigned to manage these resources, like memory, hard disk drive access, or CPU time, is the kernel of the OS. The kernel is the heart of the OS that makes everything work in harmony without conflicts or resolves them if there are any, and runs in the kernel-space. When an application is running it runs in the so-called user-space. This distinction exists to prevent application from having direct access to the underlying hardware and is enforced with the protection rings, explained before in the chapter. The running application has no knowledge of any other application being executed on the same machine and whenever it requires some resource it asks the OS through the kernel. The kernel on its behalf accesses the hard disk drive, allocates memory or executes other commands that are considered privileged and the application cannot execute. It handles all the low-level details of what the application asked and returns the results of the action.

It is a very complicated software and the most crucial part of the OS. Therefore, not every process can access the kernel directly or invoke all its functions, to avoid corruption or misuse of the low-level access it has, to gain access where one should not. This limited interface to the kernel, a sort of protection mechanism, is called system call. The details of

making a system call depend on the OS.

Programming on a high-level language usually does not involve making system calls directly. Most languages have implemented wrappers for making a system call and simplifying the system call interface. Regardless that fact, the application will eventually have to make a system call to access some of the systems resources.

2.4 Related Work

The Introspection concept gave birth to numerous interesting solutions, which target a more critical issue of the information world, that of computer security. Following are only some of the solutions produced so far. Although the approach on each research is different, the result and method of employment can potentially classify them according th the following categorization, also suggested in [9].

2.4.1 In-VM-Based Monitoring

These solutions implement part of the application inside the VM. They employ an inside agent to gather information on the VM execution state and use the elevated privileges of the hypervisor to protect the agent from corruption or subversion. Depending on the application we can refine the classification more to detection, prevention and recovery solutions. Working in a VM to gather information for the hypervisor can become a very intensive task increasing the performance overhead. The hypervisor, as well as every VM, is a complete OS, running its processes and applications, its own scheduler and intercepting its own interrupts. Besides that, there is an extra overhead, when the execution switches between a VM and the hypervisor and vice versa, a pair of events called VM-exit and VM-entry (figure 2.9). Having a monitoring and logging application on the hypervisor, triggers a considerable number of VM-exit events. This is a problem some of the following solutions tried to address by using different approaches.

Detection

To prevent this overhead, a monitoring solution, SIM [10], used the hypervisor the following way. The hypervisor, since it provides all the resource allocation, can mark the memory pages allocated to a VM, different than the guest OS would. It can mark a page read-only

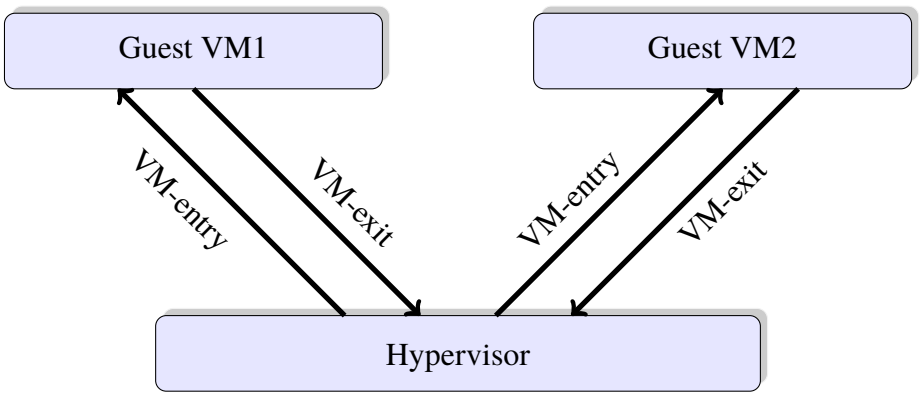


Figure 2.9. VM-exit and VM-entry events

when the OS marks it as read/write. This will trigger a VM-exit event and the hypervisor can act according to a different policy than that of the VM's OS. So, SIM, is placed inside the VM, monitoring the guest OS, but at the same time is protected by being placed on a protected by the hypervisor region of the VM's address space.

Virtuoso [11], is a tool that tries to bridge that sematic gap by automating the process of extracting OS kernel information, relevant to introspection. It runs a helper program inside the VM, which yields the wanted result. It analyzes the execution trace of that helper program and generates the introspection code that will give the same result when ran from the hypervisor. This method helps gain some knowledge about the internal machine state without having the required intricate knowledge of OS internals, but from the hypervisor's point of view.

Prevention

In [12], in the same manner, Lares tries to modify the guest OS minimally, so that the code used for monitoring can be protected easily, while all the introspection and decision making code is placed in a security VM. The two communicate through the hypervisor, which protects the hooked code in the untrusted VM, while at the same time provides information to the security VM. It also provides communication between the VMs, so that the decision making on the security VM can be enforced to the untrusted one. In this case, the monitoring happens on process creation, allowing or denying the execution of programs, as defined in a whitelist.

SHype [13] is a modified hypervisor that implements Mandatory Access Control (MAC) on shared resources between VMs. SHype is used also in [14], to provide a more fine-grained MAC on data flow between VMs and services. Hyperlink [15] implements a hybrid of protected in-VM monitoring alongside MAC-based hypervisor protection, for guest VM and hypervisor protection.

InkTag [16] introduces many different new concepts to run High Assurance Processes (HAP) in an untrusted OS. The threat model for this approach is more advanced and sophisticated. Inktag to protect the HAP employs many different mechanisms, on various levels, to ensure that there is no data leak and malicious intervention during the HAPs runtime.

Paraverification, is the concept introduced, where the kernel is required to perform some extra tasks, to provide the hypervisor high-level information about the process state. This way, the hypervisor can easily determine the high-level effects of low-level actions. Furthermore, the HAP does not interact directly with the kernel. This is done by an untrusted trampoline code, which is responsible for making the system calls instead of the HAP, and receiving the system call results from the OS, and after validating them, return them to the HAP.

To protect the contents of the HAPs memory address space, InkTag employs two EPTs. One for use during untrusted execution, which is visible by the untrusted OS, and one for use during trusted execution which is visible and used only by the hypervisor. In addition, if a page from the HAPs address space needs to be evicted, InkTag hashes the contents and encrypts them before they get written on the disk. This way it provides protection against malicious modification and access. Also, to further protect the HAP and its files, a different access control mechanism is used. Each process and file is followed by attributes, which are used to enforce an access policy, such that it will protect the files, the processes and their spawned processes. InkTag also uses a different convention to address memory and files, with the use of Object Identifiers (OI), an internal representation visible and known only to the HAP and the hypervisor. These are used to define the permissions each HAP has. Finally, InkTag modifies the actual media layout, to inject file metadata, which are used to provide crash consistency. These metadata are not visible by the untrusted OS, since these sectors are not included in the media view of the OS.

Although InkTag provides many assurances for the secure execution of a HAP, the need

to recompile applications so that they can run securely, poses a significant drawback and compromise of usability.

On a similar approach, Overshadow [17] provides a one-to-many memory mapping from the VM to physical memory, as well as other mechanisms to further protect the applications and their data. As a high-level overview, the actual data in memory depend on the process trying to access them. The contents get encrypted and hashed for untrusted processes and decrypted when the trusted application tries to access them.

To manage secure application execution inside a compromised OS Haven [18] takes a different approach. To protect the application Haven employs Intel's Software Guard Extensions (SGX). SGX allows a process to define a secure region of address space, called enclave. What Haven does, is to put the whole application in an enclave and uses an in-enclave library OS for the interactions with the OS.

On the downside, InkTag and Haven were attacked in [19] with the use of controlled-channel attacks, resulting to the extraction of substantial amounts of sensitive information from protected applications. Complete text documents were extracted, as well as outlines of JPEG images, showing that data protection during process, in not a trivial task.

2.4.2 Out-of-VM-Based Monitoring

Having a monitoring tool on the hypervisor has its benefits. At the same time though, there is a significant drawback. Although everything is visible from the hypervisors perspective, the data collected miss context. It is extremely difficult, by analyzing memory and CPU register values to understand the context, under which every execution cycle happens. Following, we comment some of the out-of-VM solutions. Some of them work on raw collected data, while others try to bridge that semantic gap to better understand the high-level commands being executed in the VM.

Detection

ReVirt [20], is a logging application. By using the hypervisor's VM access, it creates extensive logs of a VM's execution. Since the hypervisor has unlimited access to the state of the VM, ReVirt can collect and record enough information to be able to recreate and simulate the execution of the target machine. This can be very valuable for collecting

malware activity data, even after the system has been compromised, hijacked or even replaced. The replay data can prove very useful in the malware analysis field, as every non-deterministic action of a malware is recorded and deterministic results can be recreated, providing this way a full system view and impact of the malware, on every step of the malicious activity.

[21] uses the ability of the hypervisor to transparently access the running VM's internal state to collect system-level provenance, starting even from the moment the VM starts booting.

On a different approach, [22] implements a mechanism to detect insider threats. It uses VMI to stealthy monitor the user's actions and detect suspicious activity that correlates to an insider threat. Although this alert mechanism is very useful, especially due to its transparency, the insider finally gets access to the information he wants.

When the introspection idea was conceived [2], it was utilized to create a hybrid Intrusion Detection System (IDS). By placing an IDS solution on the hypervisor, it gained the best of both worlds, Host IDS (HIDS) and Network IDS (NIDS). Since it is placed outside the VM, it has the advantage of not being prone to detection, attack and corruption or evasion. It can monitor directly the network traffic, given that the Network Interface Card (NIC) is a common shared resource. On the other hand, by having the hypervisor's introspection capability, it can act also as a HIDS, by monitoring the actual system behavior and execution.

Other solutions have been proposed to fill the semantic gap between the hypervisor and the guest VM like Strider Ghostbuster [23], PoKeR [24] and VMWatcher [25]. Although all of them employ different techniques to achieve that, but unfortunately, as later researches mention [26], they fail at a point, implying that way that this semantic gap is difficult to bridge.

Prevention

This semantic gap was also addressed in [27] with a technique called process out-grafting. This method, instead of monitoring the VM as a whole, it focuses on each separate process, for a more fine-grained execution monitoring. This is done by implementing two new techniques. The first is called on-demand grafting, which can relocate a running process from the guest target VM to a security VM. This effectively bridges completely the

semantic gap, as for all intents and purposes the process is running on the same system as the monitor. This way the monitor can intercept all instructions executed by the suspicious process, without the need of hypervisor intervention. The second technique called split execution, makes a logical separation on the execution of instructions. If the process runs in user-space, it continues to run on the security VM. When there is a kernel request, like a system-call, it executes that instruction on the target VM. That technique creates an isolation between the monitor and the suspicious process, since they don't run on the same kernel, while at the same time the suspect's process perspective, it is still running inside the target VM.

Table 2.1. Overview of solutions

Solution	In-VM	Out-of-VM	Detection	Prevention	File Protection	
					Detection	Prevention
SIM [10]	X		X			
Virtuoso [11]	X		X			
Lares [12]	X			X		
SHype [13]	X			X		
InkTag [16]	X			X		
Overshadow [17]	X			X		
Haven [18]	X			X		
ReVirt [20]		X	X			
[21]		X	X			
[22]		X	X		X	
VMI [2]		X	X		X	
Strider Ghostbuster [23]		X	X		X	
PoKeR [24]		X	X		X	
VMWatcher [25]		X	X		X	
[27]		X		X		
SecVisor [28]		X		X		
HUKO [29]		X		X		
Sentry [30]		X		X		
[32]		X		X		X
Paladin [31]		X		X		X

Furthermore, SecVisor [28] and HUKO [29], propose a kernel integrity method, that protects the kernel against code injection, such that happens from rootkits. In this case, SecVisor and HUKO are part of the hypervisor. They are used to allow user allowed code execution, while at the same time prevents malicious code execution.

Sentry [30], does a more granular kernel protection by preventing low-trust kernel components from altering security-critical data used by the kernel to manage the system and itself. It protects dynamically allocated memory, is isolated from the untrusted kernel by running on the Hypervisor and reduces the overhead by monitoring only the kernel related memory pages for suspicious activity.

Paladin [31] introduces first the concept of Out-of-Guest ACL, although at a granular level, by enforcing generic access permissions. A more direct approach to filesystem integrity is presented in [32]. The author tries to protect the OS from accessing maliciously modified files. The target VM is deployed offline and all the files are signed digitally using a private key. The digests are stored on the hypervisor. When the process has been completed for all the files to be protected, the VM gets online. During its execution, whenever a file is accessed, before it gets loaded into memory, the system retrieves its digest and compares it to the copy on the hypervisor. If the file hasn't changed access or execution continues, otherwise denied.

2.1 shows a representation of the key features of the solutions presented above.

CHAPTER 3:
Design and Implementation

In this chapter we will firstly discuss the specifications, threat model and goals of this research. We will expand on the design philosophy and go in depth on the implementation.

3.1 Specifications

As far as we know, all works on VM monitoring and security focus in kernel and OS protection, malicious activity monitoring or extensive logging for replay and online or offline forensic purposes, or secure resource sharing among VMs. All these solutions do not provide any protection for the actual files of the system, which can be maliciously accessed when the VM has been compromised. VM security is a very active research filed, that produces many solutions, each with different focus, but generally surrounding the malware protection realm. Many of these are referenced in [9], an extensive survey on hypervisor-based solutions.

In this research, we will try to leverage the Xen's VMI capabilities and create a mechanism to protect some critical files on a VM. We want to create an alternate ACL on the hypervisor, that will include modified permissions for file access. The hypervisor will monitor what files are being accessed and cross-check the action with the ACLs entries, enforcing the out-of-guest ACL. Although a similar approach was employed with Paladin [31], and integrity is improved in [32], there are some fundamental differences.

We will focus on the use of type-I hypervisor instead of type-II. Moreover, we want the guest OS to be unmodified and without any code, app or monitor injection that must be protected. We will employ the stealthy property of DRAKVUF [4], to make the process of file protection completely transparent to the guest OS, retaining this way a zero-footprint monitor on the guest. DRAKVUF also helps in bridging the semantic gap between the hypervisor and the VM with the use of a Rekall profile [33], having this way access to selected kernel structures. Furthermore, we want to employ a per user ACL, enforced on specific files or whole folders, sometimes not essential to the OS. Essentially, we want to protect any type of data, regardless of the content. Confidentiality is enforced by denying

even read access, while integrity by denying write. This mechanism must also extend to the root user, since our threat model assumes that the system is compromised. Finally, we will try to enforce a specific file access mode, where log files are forced to open always in append mode rather than write, regardless of the program, process or user accessing them. To achieve all that we will intercept all relevant system calls and verify the validity of the request.

CHAPTER 4:
Results and Conclusion

In this chapter, the information of the relevant software and hardware is presented. In the first section a short introduction on virtualization and its benefits is given. Then the types of hypervisors and Xen, the platform we will work on, are described. The next section refers to VMI and Xen's capabilities in that field, the LibVMI API and DRAKVUF, the library and main application we will leverage, as well as the system call functionality and convention. Finally, some of the existing solutions that leverage introspection are reviewed.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 5:
Future Work

In this chapter, the information of the relevant software and hardware is presented. In the first section a short introduction on virtualization and its benefits is given. Then the types of hypervisors and Xen, the platform we will work on, are described. The next section refers to VMI and Xen's capabilities in that field, the LibVMI API and DRAKVUF, the library and main application we will leverage, as well as the system call functionality and convention. Finally, some of the existing solutions that leverage introspection are reviewed.

THIS PAGE INTENTIONALLY LEFT BLANK

List of References

[1] P. Mell, T. Grance *et al.*, "The nist definition of cloud computing," 2011.

[2] T. Garfinkel, M. Rosenblum *et al.*, "A virtual machine introspection based architecture for intrusion detection." in *Ndss*, 2003, vol. 3, pp. 191–206.

[3] B. D. Payne, "Libvmi," Sandia National Laboratories, Tech. Rep., 2011.

[4] T. K. Lengyel, S. Maresca, B. D. Payne, G. D. Webster, S. Vogl, and A. Kiayias, "Scalability, fidelity and stealth in the drakvuf dynamic malware analysis system," in *Proceedings of the 30th Annual Computer Security Applications Conference*, 2014.

[5] M. Rosenblum and T. Garfinkel, "Virtual machine monitors: Current technology and future trends," *Computer*, vol. 38, no. 5, pp. 39–47, 2005.

[6] "Xen project software overview," accessed: 2017-05-09. Available: https://wiki.xenproject.org/wiki/Xen_Project_Software_Overview

[7] B. D. Payne, D. d. A. Martim, and W. Lee, "Secure and flexible monitoring of virtual machines," in *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*. IEEE, 2007, pp. 385–397.

[8] K. Lars, "Virtual machine introspection: A security innovation with new commercial applications," Aug 2016, accessed: 2017-05-09. Available: https://www.linux.com/news/virtual-machine-introspection-security-innovation-new-commercial-applications

[9] E. Bauman, G. Ayoade, and Z. Lin, "A survey on hypervisor-based monitoring: approaches, applications, and evolutions," *ACM Computing Surveys (CSUR)*, vol. 48, no. 1, p. 10, 2015.

[10] M. I. Sharif, W. Lee, W. Cui, and A. Lanzi, "Secure in-vm monitoring using hardware virtualization," in *Proceedings of the 16th ACM conference on Computer and communications security*. ACM, 2009, pp. 477–487.

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

[12] B. D. Payne, M. Carbone, M. Sharif, and W. Lee, "Lares: An architecture for secure active monitoring using virtualization," in *Security and Privacy, 2008. SP 2008. IEEE Symposium on*. IEEE, 2008, pp. 233–247.

[13] R. Sailer, T. Jaeger, E. Valdez, R. Caceres, R. Perez, S. Berger, J. L. Griffin, and L. Van Doorn, "Building a mac-based security architecture for the xen open-source hypervisor," in *Computer security applications conference, 21st Annual*. IEEE, 2005, pp. 10–pp.

[14] B. Hay and K. Nance, "Forensics examination of volatile system data using virtual introspection," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 3, pp. 74–82, 2008.

[15] J. Xiao, L. Lu, H. Wang, and X. Zhu, "Hyperlink: Virtual machine introspection and memory forensic analysis without kernel source code," in *Autonomic Computing (ICAC), 2016 IEEE International Conference on*. IEEE, 2016, pp. 127–136.

[16] O. S. Hofmann, S. Kim, A. M. Dunn, M. Z. Lee, and E. Witchel, "Inktag: Secure applications on an untrusted operating system," in *ACM SIGARCH Computer Architecture News*, no. 1. ACM, 2013, vol. 41, pp. 265–278.

[17] X. Chen, T. Garfinkel, E. C. Lewis, P. Subrahmanyam, C. A. Waldspurger, D. Boneh, J. Dwoskin, and D. R. Ports, "Overshadow: a virtualization-based approach to retrofitting protection in commodity operating systems," in *ACM SIGARCH Computer Architecture News*, no. 1. ACM, 2008, vol. 36, pp. 2–13.

[18] A. Baumann, M. Peinado, and G. Hunt, "Shielding applications from an untrusted cloud with haven," *ACM Transactions on Computer Systems (TOCS)*, vol. 33, no. 3, p. 8, 2015.

[19] Y. Xu, W. Cui, and M. Peinado, "Controlled-channel attacks: Deterministic side channels for untrusted operating systems," in *Security and Privacy (SP), 2015 IEEE Symposium on*. IEEE, 2015, pp. 640–656.

[20] G. W. Dunlap, S. T. King, S. Cinar, M. A. Basrai, and P. M. Chen, "Revirt: Enabling intrusion analysis through virtual-machine logging and replay," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 211–224, 2002.

[21] P. Macko, M. Chiarini, M. Seltzer, and S. Harvard, "Collecting provenance via the xen hypervisor." in *TaPP*, 2011.

[22] M. Crawford and G. Peterson, "Insider threat detection using virtual machine introspection," in *System Sciences (HICSS), 2013 46th Hawaii International Conference on*. IEEE, 2013, pp. 1821–1830.

[23] Y.-M. Wang, D. Beck, B. Vo, R. Roussev, and C. Verbowski, "Detecting stealth software with strider ghostbuster," in *Dependable Systems and Networks, 2005. DSN 2005. Proceedings. International Conference on*. IEEE, 2005, pp. 368–377.

[24] R. Riley, X. Jiang, and D. Xu, "Multi-aspect profiling of kernel rootkit behavior," in *Proceedings of the 4th ACM European conference on Computer systems*. ACM, 2009, pp. 47–60.

[25] X. Jiang, X. Wang, and D. Xu, "Stealthy malware detection through vmm-based out-of-the-box semantic view reconstruction," in *Proceedings of the 14th ACM conference on Computer and communications security*. ACM, 2007, pp. 128–138.

[26] C. Mahapatra and S. Selvakumar, "An online cross view difference and behavior based kernel rootkit detector," *ACM SIGSOFT Software Engineering Notes*, vol. 36, no. 4, pp. 1–9, 2011.

[27] D. Srinivasan, Z. Wang, X. Jiang, and D. Xu, "Process out-grafting: an efficient out-of-vm approach for fine-grained process execution monitoring," in *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 2011, pp. 363–374.

[28] A. Seshadri, M. Luk, N. Qu, and A. Perrig, "Secvisor: A tiny hypervisor to provide lifetime kernel code integrity for commodity oses," in *ACM SIGOPS Operating Systems Review*, no. 6. ACM, 2007, vol. 41, pp. 335–350.

[29] X. Xiong, D. Tian, P. Liu *et al.*, "Practical protection of kernel integrity for commodity os from untrusted extensions." in *NDSS*, 2011, vol. 11.

[30] A. Srivastava and J. Giffin, "Efficient protection of kernel data structures via object partitioning," in *Proceedings of the 28th annual computer security applications conference*. ACM, 2012, pp. 429–438.

[31] A. Baliga, L. Iftode, and X. Chen, "Automated containment of rootkits attacks," *Computers & Security*, vol. 27, no. 7, pp. 323–334, 2008.

[32] M. R. Nasab, "Security functions for virtual machines via introspection," 2012.

[33] "Wrekall memory forensic framework." Available: http://www.rekall-forensic.com/

THIS PAGE INTENTIONALLY LEFT BLANK

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California