NISTIR 8221

A Methodology for Enabling Forensic Analysis Using Hypervisor Vulnerabilities Data

Ramaswamy Chandramouli
Anoop Singhal
Duminda Wijesekera
Changwei Liu

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8221

NIST
National Institute of
Standards and Technology
U.S. Department of Commerce

NISTIR 8221

A Methodology for Enabling Forensic Analysis Using Hypervisor Vulnerabilities Data

Ramaswamy Chandramouli
Anoop Singhal
Duminda Wijesekera
Computer Security Division
Information Technology Laboratory

Changwei Liu
George Mason University
Fairfax, VA

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8221

June 2019

U.S. Department of Commerce
Wilbur L. Ross, Jr., Secretary

National Institute of Standards and Technology
Walter Copan, NIST Director and Under Secretary of Commerce for Standards and Technology

National Institute of Standards and Technology Interagency or Internal Report 8221
35 pages (June 2019)

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8221

Certain commercial entities, equipment, or materials may be identified in this document in order to describe an experimental procedure or concept adequately. Such identification is not intended to imply recommendation or endorsement by NIST, nor is it intended to imply that the entities, materials, or equipment are necessarily the best available for the purpose.

There may be references in this publication to other publications currently under development by NIST in accordance with its assigned statutory responsibilities. The information in this publication, including concepts and methodologies, may be used by federal agencies even before the completion of such companion publications. Thus, until each publication is completed, current requirements, guidelines, and procedures, where they exist, remain operative. For planning and transition purposes, federal agencies may wish to closely follow the development of these new publications by NIST.

Organizations are encouraged to review all draft publications during public comment periods and provide feedback to NIST. Many NIST cybersecurity publications, other than the ones noted above, are available at https://csrc.nist.gov/publications.

Comments on this publication may be submitted to:

National Institute of Standards and Technology
Attn: Computer Security Division, Information Technology Laboratory
100 Bureau Drive (Mail Stop 8930) Gaithersburg, MD 20899-8930
Email: nistir8221@nist.gov

All comments are subject to release under the Freedom of Information Act (FOIA).

Reports on Computer Systems Technology

The Information Technology Laboratory (ITL) at the National Institute of Standards and Technology (NIST) promotes the U.S. economy and public welfare by providing technical leadership for the Nation's measurement and standards infrastructure. ITL develops tests, test methods, reference data, proof-of-concept implementations, and technical analyses to advance the development and productive use of information technology. ITL's responsibilities include the development of management, administrative, technical, and physical standards and guidelines for the cost-effective security and privacy of other than national security-related information in federal information systems.

Abstract

Hardware/Server Virtualization is a foundational technology in a cloud computing environment and the hypervisor is the key software in that virtualized infrastructure. However, hypervisors are large pieces of software with several thousand lines of code and are therefore known to have vulnerabilities. Hence a capability to perform forensic analysis to detect, reconstruct and prevent attacks based on recent vulnerabilities on an ongoing basis is a critical requirement in cloud environments. The purpose of this document is to develop a methodology to enable this forensic analysis. Two open-source hypervisors—Xen and Kernel-based Virtual Machine (KVM)-were chosen as platforms to illustrate the methodology, and the source for vulnerability data is the National Institute of Standards and Technology's National Vulnerability Database (NIST-NVD). The vulnerabilities were classified in terms of hypervisor functionality, attack type, and attack source. Based on the relative distribution of vulnerabilities in a hypervisor functionality, two sample attacks were launched to exploit vulnerabilities in the target hypervisor functionality, and the associated system calls were logged. The gaps in evidence data that is required for fully detecting and reconstructing those attacks are identified and techniques required to gather missing evidence are incorporated during subsequent attack runs. This is intended to be an iterative process.

Keywords

cloud computing; forensic analysis; hypervisors; Kernel-based Virtual Machine (KVM); vulnerabilities; Xen

Acknowledgments

The authors thank Ms. Isabel Van Wyk for her valuable editorial review.

Audience

The target audience for this document includes security staff and Chief Information Security Officers (CISO) in virtualized infrastructures used for enterprise computing needs or for offering cloud services.

Trademark Information

Executive Summary

Server/Hardware Virtualization is now an established technology in cloud computing environments as it enables ubiquitous access to shared pools of system resources and high-level services provisioned with minimal management effort. The key enabling software of this technology is the hypervisor. This software layer lies between the physical hardware (called the hypervisor host) and multiple application workloads executing in Virtual Machines (VMs or guest machines). The hypervisor supports the guest machines by presenting to their operating systems (OSs) a virtual hardware platform and managing their execution.

However, hypervisors are large pieces of software with many lines of code and known vulnerabilities. Hence a capability to perform forensic analysis to detect, reconstruct and prevent attacks based on recent vulnerabilities on an ongoing basis is a critical requirement in cloud environments. The purpose of this document is to develop a methodology to enable this forensic analysis.

The first step of the methodology is to analyze the recent vulnerabilities and attacks on hypervisor products. Two open-source hypervisors - Xen and Kernel-based Virtual Machine (KVM)-were chosen as platforms to illustrate the methodology, and the source for vulnerability data is the National Institute of Standards and Technology's National Vulnerability Database (NIST-NVD). The vulnerabilities were classified based on three categories – the hypervisor functionality where the vulnerability exists, attack type and attack source. The outcome of this step is to obtain the relative distribution of recent hypervisor vulnerabilities for the two products in the three categories.

The second step of the methodology is to identify the hypervisor functionality that is most impacted and to build sample attacks. These sample attacks were run for the chosen target hypervisor functionality, and the associated system calls were logged. The third step is an iterative process to determine the gaps in evidence data that is required for fully detecting and reconstructing those attacks and to identify techniques required to gather the needed evidence during subsequent attack runs.

The intended benefit of the methodology is to enable all stakeholders (cloud providers and customers) to gain a better understanding of recent hypervisor vulnerabilities and attack trends, identify forensic information needed to reveal the presence of such attacks, and develop guidance on taking proactive steps to detect and prevent those attacks in their operating environments.

Table of Contents

1 Introduction

Most cloud services are provided through a virtualized infrastructure. Since virtualization of all system resources—including processors, memory, and Input/Output (I/O) devices—makes it possible to run multiple operating systems on a single physical platform (host), virtualization is a key technology in cloud computing environments that enables ubiquitous access to shared pools of system resources and high-level services provisioned with minimal management effort [1, 2]. An Operating System (OS) directly controls hardware resources in a non-virtualized system, but virtualization, typically performed by a hypervisor (also called a virtual machine monitor or VMM) [3] within a cloud environment, provides a mechanism that abstracts the hardware and system resources from an OS. As a software layer that lies between the physical hardware and the Virtual Machines (VMs or guest machines), a hypervisor supports the guest machines by presenting the guest OSs with a virtual operating platform and managing their execution.

However, hypervisors are large pieces of software with many lines of code and known vulnerabilities [4]. While there is published research dedicated to characterizing and assessing hypervisor vulnerabilities as well as detecting and forensically analyzing the corresponding attacks [4-8], there is no formal methodology for enabling forensic analysis on open-source hypervisors, such as Kernel-based Virtual Machine (KVM) and Xen. Motivated by the work presented in [4], which characterized hypervisor vulnerabilities as of July 2012 with the objective of preventing their exploitation, this document considers the recent vulnerability reports associated with Xen and KVM in the NIST National Vulnerability Database (NIST-NVD). The objective is to analyze recent trends in hypervisor attacks, provide suggestions for mitigating hypervisor attack risks, and identify evidence of those attacks. The main contributions of this publication are as follows: (1) all vulnerabilities of the Xen and KVM hypervisor products from the 2016 and 2017 NIST National Vulnerability Database (NIST-NVD) were analyzed and classified based on their underlying functionalities, attack types, and sources of attacks; (2) classification of the above mentioned Xen and KVM hypervisor vulnerabilities based on three categories – hypervisor functionality, attack types and attack sources, and generating a relative distribution of the number of vulnerabilities (3) Use the previous data as the basis to develop sample attacks that exploit vulnerabilities in a target hypervisor functionality and run with system call logging capability, to identify coverage of the evidence needed for detecting, reconstructing and preventing these attacks.

The rest of the publication is organized as follows. Section 2 presents an overview of hypervisor taxonomy and briefly describes the architecture of Xen and KVM hypervisor products as well as related work in the area of vulnerability/forensic analysis. Section 3 discusses the gathering of recent vulnerabilities in the two hypervisor products and the classification of those vulnerabilities in terms of hypervisor functionality, attack types and attack sources. Section 4 describes the sample attacks, information gathered in the initial log configuration, and discusses the gaps in forensic evidence that is required for fully detecting and reconstructing those attacks. Identification and discussion of techniques required to gather the needed evidence and the inclusion of those techniques in subsequent attack runs are also part of this section. Section 5 provides the summary and benefits.

2 Technology Background and Related Work

This section provides an overview of hypervisor taxonomy and briefly describes the various building blocks in the architecture of two open-source hypervisor products. It also summarizes related work in the area of hypervisor vulnerability/forensic analysis.

2.1 Hypervisors

Hypervisors are software and/or firmware modules that virtualize system resources such as the Central Processing Unit (CPU), memory, and devices. In [9], Popek and Goldberg classify hypervisors as Type 1 hypervisors and Type 2 hypervisors. Type 1 hypervisors run directly on the host's hardware to control the hardware and manage guest operating systems (Guest OS). For this reason, Type 1 hypervisors are sometimes called bare metal hypervisors and include Xen, Microsoft Hyper-V, and VMware ESX/ESXi. Type 2 hypervisors are similar to other computer programs that run on an OS as a process. VMware Player, VirtualBox, Parallels Desktop for Mac, and Quick Emulator (QEMU) are Type 2 hypervisors. Some systems have features of both. For example, Linux's Kernel-based Virtual Machine (KVM) is a kernel module that effectively converts the host OS to a Type 1 hypervisor but is also categorized as a Type 2 hypervisor because Linux distributions are still general-purpose OSs with other applications competing for VM resources [10].

According to the 2015 State of Hyperconverged Infrastructure Market Report by ActualTech media [23], there are four leading products in the hypervisor market: Microsoft Hyper-V, VMware VSphere/ESX, Citrix XenServer/Xen, and KVM. The first two - Microsoft Hyper-V and VMware VSphere/ESX are commercial products, while the last two are open-source products. These open-source products (i.e., Xen and KVM) were chosen as platforms for illustrating the methodology that forms the subject of this document. Their architectures are briefly discussed below.

2.1.1 Xen

Figure 1 shows the architecture of Xen. In this design, the Xen hypervisor provides two types of domains – a single control domain (also called Domain0 or Dom0) and multiple guest domains (also called DomainU or DomU). Since the hypervisor supports two different virtualization modes, Paravirtualization (PV) and Hardware-assisted Virtualization (HVM) [11], a total of three different types of VMs – Domain0 VM (Dom0-VM), DomainU--PVM (DomU-PVM) and DomainU--HVM (DomU-HVM) can be hosted on the Xen platform. Dom0 is the initial domain started by the Xen hypervisor on booting up a privileged domain that plays the administrator role and supplies services for DomU VMs. For the two kinds of DomU guests, PV is a highly efficient and lightweight virtualization technology introduced by Xen in which Xen PV does not require virtualization extensions from the host hardware. Thus, PV enables virtualization on hardware architectures that do not support HVM, but it requires PV-enabled kernels and PV drivers to power a high performance virtual server. HVM requires hardware extensions, and Xen typically uses QEMU, a generic hardware emulator [15], for simulating PC hardware (e.g., CPU, Basic Input Output System (BIOS), Integrated Drive Electronics (IDE), Video Graphics Array (VGA), network cards, and Universal Serial Bus (USBs)). Because most I/O and network operations in HVM mode are done by using simulation technologies, performance of DomU-HVMs are inferior to DomU-PVMs[28]. Xen 4.4 provides a new virtualization mode named Para Virtualized

Hardware (PVH). PVH guests are lightweight HVM-like guests that use virtualization extensions in the host hardware. Unlike HVM guests, instead of using QEMU to emulate devices, PVH guests use PV drivers for I/O and native OS interfaces for virtualized timers and virtualized interrupts. PVH guests require PVH-enabled guest OS [11].

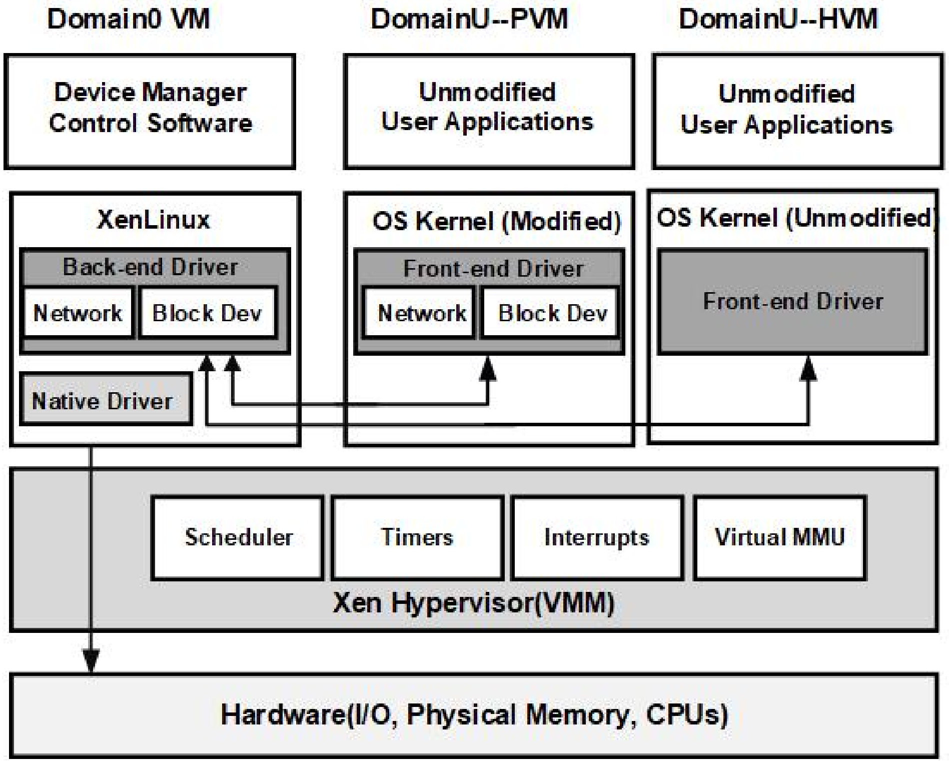


Figure 1: The Xen architecture

2.1.2 KVM

In the open-source hypervisor projects, the Kernel-based Virtual Machine (KVM) is a relatively new product which was first introduced in 2006 and soon merged into the Linux kernel (2.6.20). KVM is a full virtualization solution for Linux on x86 hardware containing virtualization extensions (Intel VT or AMD-V) where VMs run as normal Linux processes [12]. Figure 2 shows the KVM architecture, in which the KVM module uses QEMU to create guest VMs running as separate user processes. Because KVM is installed on top of the host OS, it is considered a Type 2 hypervisor. However, the KVM kernel module turns the Linux kernel into a Type 1 bare-metal hypervisor, providing the power and functionality of even the most complex and powerful Type 1 hypervisors.

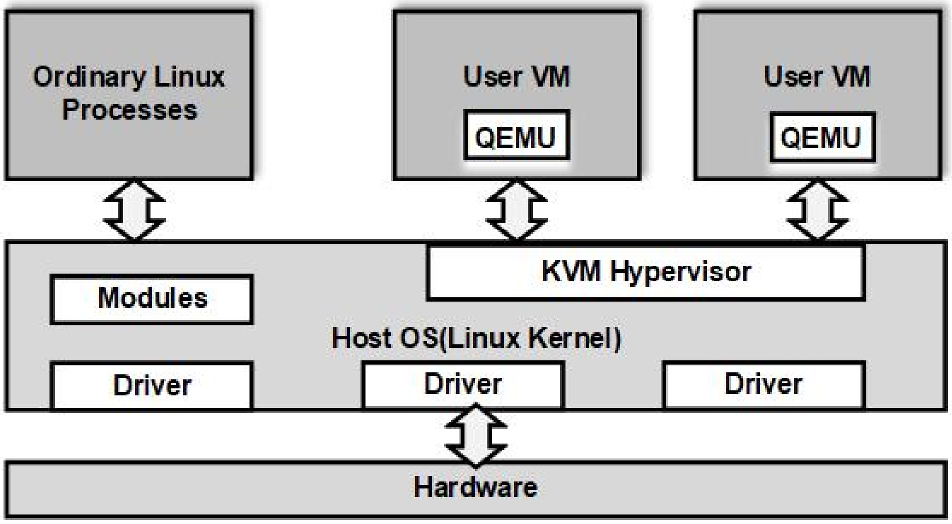


Figure 2: The KVM architecture

2.2 Related Work

Hypervisor attacks are categorized as external attacks and defined as exploits of the hypervisor's vulnerabilities which allow attackers to gain accessibility and authorization over the hypervisors [13]. In support of hypervisor defense, Perez-Botero et al. characterized Xen and KVM vulnerabilities based on hypervisor functionalities in 2012 [4]. However, these vulnerabilities cannot be used as the basis for characterizing many recent attacks. Using the NIST 800-115 security testing framework, Thongtua et al [5] assessed the vulnerabilities of widely used hypervisors, including VMware ESXi, Citrix XenServer, and KVM then performed some sample experiments in order to derive severity scores, and attack impacts. In an effort to develop hypervisor forensic methods, researchers discussed the attacks on hypervisors, their forensic mechanisms and challenges [8], and leveraged existing memory forensic techniques to perform forensic analysis on hypervisor attacks [7].

3 Classification of Hypervisor Vulnerabilities

This section describes the first step of our methodology which involves the collection of recent vulnerabilities in Xen and KVM hypervisor products and classifying them based on three categories-Hypervisor Functionality, Attack Types and Attack Sources.

A brief description of the information sources that were used and the steps adopted as part of the approach for obtaining the relative distribution of vulnerabilities is given in Sections 3.1, 3.2, and 3.3.

3.1 The Hypervisor Vulnerabilities Data in the NIST-NVD

The NIST-NVD is the U.S. government repository of standards-based vulnerability management data and includes databases of security checklist references, security-related software flaws, misconfigurations, product names, and impact metrics [14]. The first task of the first step is to obtain all vulnerabilities (tagged with Common Vulnerabilities and Exposures (CVE) numbers)) in two open-source hypervisors (Xen and KVM) from the NIST-NVD for the years 2016 and 2017.

As listed in Appendix A, a search of the NIST-NVD for the vulnerabilities based on keyword "Xen" and "KVM" posted during the years 2016 and 2017 revealed 83 Xen hypervisor vulnerabilities and 20 KVM hypervisor vulnerabilities. These vulnerabilities were then classified based on the following three categories:

- Hypervisor Functionality where the vulnerability exists (attack vector channel)
- Attack Type (impact of the attack by exploiting the vulnerability)
- Attack Source (users in different virtualization component levels (e.g., guest OS, host OS or hypervisor) and different privilege levels (e.g., user, administrator) who can launch attacks if malicious)).

3.2 Classifying Vulnerabilities Based on Hypervisor Functionality

To obtain a better understanding of different hypervisor vulnerabilities in terms of impacted hypervisor functions, Perez-Botero et al. considered 11 basic functionalities that a traditional hypervisor provides and mapped vulnerabilities to them [4]. These functionalities include:

1) Virtual CPUs (vCPU)
2) Symmetric Multiprocessing (VSMP)
3) Soft Memory Management Unit (MMU)
4) I/O and Networking
5) Paravirtualized I/O
6) Interrupt and Timer mechanisms
7) Hypercalls
8) VMExit
9) VM Management
10) Remote Management Software
11) Hypervisor Add-ons

In the approach adopted in this document, we used the above 11 functions in a slightly modified way. The functionalities 4 and 5 were merged into a single functionality based on the fact that they both pertain to I/O. (A detailed description of all these functionalities can be found in Appendix B). All reported Xen and KVM vulnerabilities during the years 2016 and 2017 were mapped to these hypervisor functionalities based on the approach in [4], which are listed in Appendix A. A brief description of sample vulnerabilities associated with functionalities is given in Table 1:

Table 1: A sample vulnerability for each hypervisor functionality

Hypervisor Functionality	Sample Vulnerability
vCPU	CVE-2017-10923 is an example of vCPU vulnerability in which Xen through 4.8.x does not validate a vCPU array index upon sending a software generated interrupt (SGI), which allows a guest OS user to cause a denial-of-service (DoS) attack, finally resulting in crashing the hypervisor.
VSMP	NONE
Soft MMU	An example of soft MMU vulnerability is CVE-2017-17565, which existed up to Xen version 4.9.x. Due to an incorrect assertion related to machine-to- people (M2P), this vulnerability allows a paravirtualized guest OS user to cause a DoS attack when both the shadow mode and log-dirty mode are set up and working.
I/O and Networking	CVE-2017-15589 is an example of an I/O and networking vulnerability discovered in Xen versions through 4.9.x which allows x86 HVM guest OS users to obtain sensitive information from the host OS (or an arbitrary guest OS). In these versions of Xen, at least one write path was found wherein the data that had been stored in an internal structure could contain bits from an uninitialized hypervisor stack slot. A subsequent emulated read would retrieve these bits.
Interrupt/Timer	CVE-2018-7542 is an example of an interrupt/timer vulnerability caused by leveraging the mishandling of configurations that lack a local Advanced Programmable Interrupt Controller (APIC). It was discovered in Xen 4.8.x through 4.10.x. This vulnerability allows an x86 PVH guest OS user to cause a DoS attack (a NULL pointer dereference and hypervisor crash).
Hypercalls	An example of hypercall vulnerability is CVE-2017-8903, which is reported through Xen 4.8.x on 64-bit platforms that might allow a PV guest OS user to execute arbitrary code on the host OS by mishandling page tables after an IRET hypercall.
VMExit	The exploit of a VM Exit-handling code usually leads to a DoS attack. An example of a VMExit vulnerability is CVE-2017-2596, in which the “nested_vmx_check_vmptr” function in arch/x86/kvm/vmx.c in the Linux kernel through 4.9.8 improperly emulates the VMXON instruction that puts the processor in Virtual Machine Extensions (VMX) root mode. This then

Hypervisor Functionality	Sample Vulnerability
	allows a KVM L1 guest OS user to cause a DoS attack (the host OS memory consumption) by leveraging the mishandling of page references.
VM Management	CVE-2016-4963 is an example vulnerability of VM management. The libxl device-handling in Xen through 4.6.x allows local guest OS users access to the driver domain to cause a denial of service (management tool confusion) by manipulating information in the backend directories in xenstore.
Remote Management Software	The exploit of the management functionality may allow a host compromise. An example of VM management functionality vulnerability is CVE-2016-5302. When a deployment has been upgraded from an earlier release, XenServer 7.0 before the vendor's Hot x XS70E003 may allow a remote attacker on the management network to compromise a host by leveraging credentials for an active directory account.
Hypervisor Add-ons	CVE-2016-0749 is an example vulnerability of hypervisor add-ons. By leveraging the smartcard interaction in Simple Protocol for Independent Computing Environments (SPICE) as KVM add-ons, a remote attacker can cause a DoS attack (QEMU-KVM process crash) or possibly execute arbitrary code via vectors related to connecting to a guest VM, which triggers a heap-based buffer overflow.

3.3 Obtaining Relative Distribution of Vulnerabilities

After tagging each vulnerability with the three categories listed above, the number of vulnerabilities were summarized for each category and percentages of occurrence within each category were computed to obtain their relative distribution.

Table 2: The vulnerabilities of Xen and KVM classified by hypervisor functionality

Number	Hypervisor Functionality	Xen (Number & Percentage of the Total)	KVM (Number & Percentage of the Total)
1	vCPU	6 (7 %)	4 (20 %)
2	VSMP	0 (0 %)	0 (0 %)
3	Soft MMU	34 (40 %)	5 (25 %)
4	I/O and Networking	24 (29 %) Five are fully-virtualized; 19 are paravirtualized; none are direct access or self-virtualized.	4 (20 %) All are fully-virtualized.
5	Interrupt/Timer	7 (8 %)	3 (15 %)
6	Hypercalls	3 (4 %)	1 (5 %)
7	VMExit	1 (1 %)	2 (10 %)

Number	Hypervisor Functionality	Xen (Number & Percentage of the Total)	KVM (Number & Percentage of the Total)
8	VM Management	7 (8 %)	0 (0 %)
9	Remote Management Software	1 (0 %)	0 (0 %)
10	Hypervisor Add-ons	0 (0 %)	1 (5 %)

Classifications based on the hypervisor functionalities are shown in Table 2. With the exception of the functionality of virtual symmetric multiprocessing, all functionalities were reported as having vulnerabilities. The number of vulnerabilities and the percentages within each hypervisor offering are listed. The table reveals that there are more reported Xen vulnerabilities than KVM. One of the reasons can be attributed to a broader user base for Xen. Furthermore, approximately 69 % of the vulnerabilities in Xen and 45 % of the vulnerabilities in KVM are concentrated in two functionalities—Soft MMU and I/O and Networking. A detailed reading of CVE reports reveals that these vulnerabilities primarily originated in page tables and I/O grant table emulation. Additionally, the vulnerabilities based on the I/O and Networking functionality were also associated with each of the four types of I/O virtualization: (1) fully virtualized devices, (2) paravirtualized devices, (3) direct access devices, and (4) self-virtualized devices. Table 2 shows that most of the I/O and networking vulnerabilities in Xen came from paravirtualized devices, while all I/O and networking vulnerabilities in KVM came from fully-virtualized devices. This is due to the fact that in most Xen deployments, I/O and networking functionality is configured using a paravirtualized device, while in KVM, that functionality is configured using a fully virtualized device.

Table 3: The types of attacks caused by Xen and KVM vulnerabilities

Type of Attack	Xen (Number & Percentage of the Total)	KVM (Number & Percentage of the Total)
Denial-of-service (DoS)	48 (four have other impacts) (44 %)	17 (three have other impacts) (63 %)
Privilege escalation	33 (16 have other impacts) (30 %)	3 (two have other impacts) (11 %)
Information leakage	15 (five have other impacts) (14 %)	5 (19 %)
Arbitrary code execution	8 (two have other impacts) (7 %)	2 (all have other impacts) (7 %)
Reading/modifying/deleting a file	3 (3 %)	0 (0 %)
Others including compromising a host, canceling other administrators' operations and corrupting data	3 (3 %)	0 (0 %)

Classifications based on the attack types and the sources of attacks are listed in Table 3 and Table 4, respectively. Table 3 reveals that the most common attack type was DoS (44 % for Xen and

63 % for KVM), indicating that attacking cloud services' availability could be a serious cloud security problem. The other top attacks were privilege escalation (30 % for Xen and 11 % for KVM), information leakage (14 % for Xen and 19 % for KVM), and arbitrary code execution (7 % for Xen and 7 % for KVM). Although each of these thre;e attacks occurs with less frequency than a DoS attack, they all carry the potential risk of either leaking sensitive user information or compromising the hosts or guest VMs. Table 4 shows that the greatest source of all attacks was guest OS users (76 % for Xen and 85 % for KVM). This suggests that cloud providers must closely monitor guest users' activities in order to reduce attack risks.

Table 4: Attack Sources and Number of Exploits

Source of Attack	Xen (Number & Percentage of the Total)	KVM (Number & Percentage of the Total)
Administrator	2 (Management) (2 %)	0 (0 %)
Guest OS administrator	17 (including HVM and PV administrators) (20 %)	1 (5 %)
Guest OS user	63 (including Advanced RISC Machine (ARM), X86, HVM and PV users) (76 %)	17 (including KVM L1, L2, and privileged users) (85 %)
Remote attacker	1 (1 %)	1 (including an authenticated remote guest user) (5 %)
Host OS user	0 (0 %)	1 (5 %)

4 Sample Attacks and Forensic Analysis

The second step of the methodology is to identify the hypervisor functionality that is most impacted by the vulnerabilities (from relative distribution) and use that as the basis to build sample attacks. Since numerous vulnerabilities are related to the Xen soft MMU functionality, this section illustrates two sample attacks that exploit vulnerabilities in this functionality, CVE-2017-7228 and CVE-2016-6258, to demonstrate how the requisite evidence for detecting and reconstructing hypervisor attacks is determined.

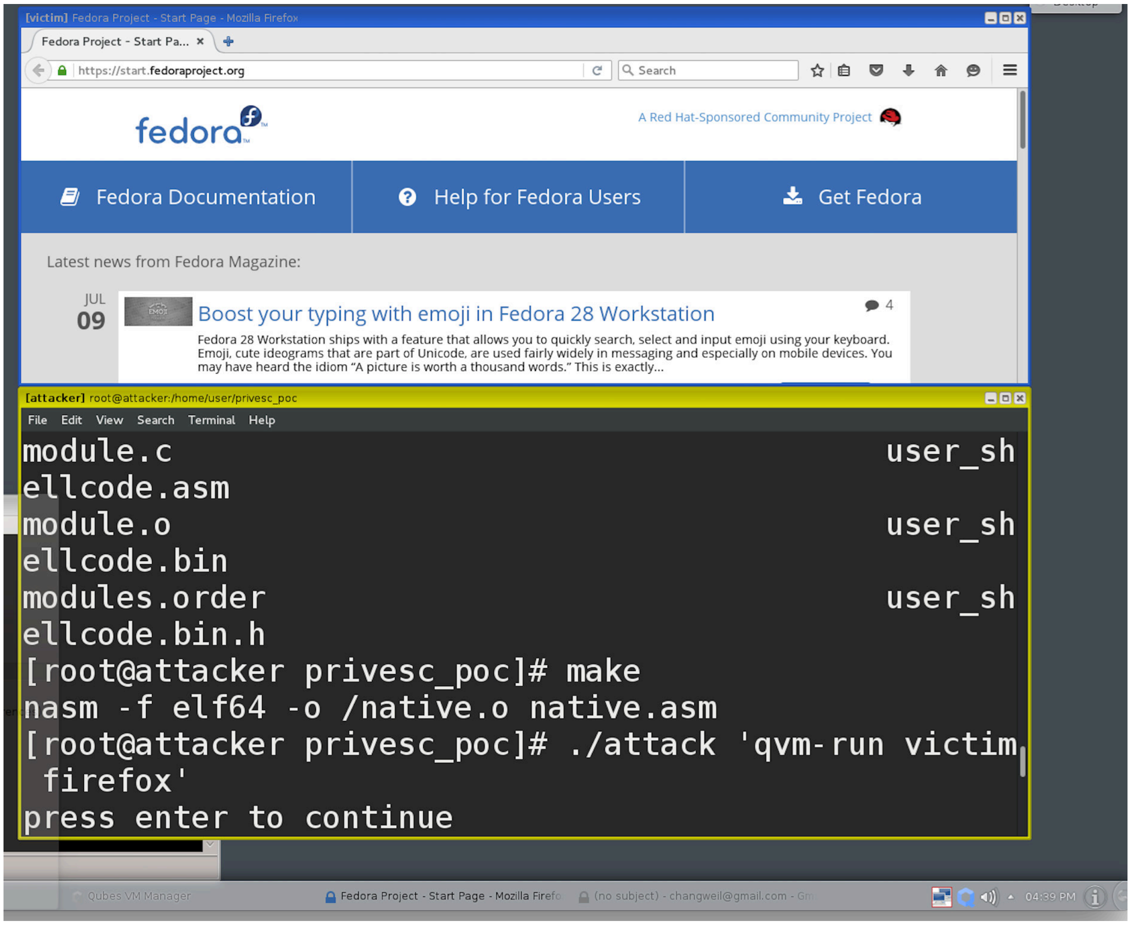
4.1 The Two Sample Attacks

As presented in Section 2.1.1., the Xen hypervisor can host three types of VMs, Dom0-VM, DomU-PVM and DomU-HVM. The PV module in the hypervisor that implements PV mode and supports DomU-PVMs has been widely utilized for its higher performance [25]. However, since this module uses complex code to emulate the MMU, it introduces many vulnerabilities, such as CVE-2017-7228 and CVE-2016-6258.

Known by Xen as XSA-212, CVE-2017-7228 was first reported by Jann Horn of Google's Project Zero in 2017 [20]. Horn discovered that this vulnerability in X86 64 bit Xen (including 4.8.x, 4.7.x, 4.6.x, 4.5.x, and 4.4.x versions) was caused by an insufficient check on the function "XENMEM_exchange", which allows the PV guest user as the function caller to access hypervisor memory outside of the PV guest VM's provisioned memory. Therefore, a malicious 64-bit PV guest who can make a hypercall "HYPERVISOR_memory_op" function to invoke the "XENMEM_exchange" function may be able to access all of a system's memory. The consequent VM escape from DomU to Dom0 (the process of breaking out of a guest VM and interacting with the hypervisor's host operating system) can enable the PV guest user to cause hypervisor host crash and information leakage. The resulting increase in privilege can also enable the malicious PV guest user to execute commands like "qvm-run victim firefox" to open a Firefox web-browser in the "victim" guest VM, which can only be executed by Dom0 as shown in Figure 3.

CVE-2016-6258 is also known as XSA-182, which was reported by Jeremie Boutoille from Quarklab in 2016 [21]. In the PV module, page tables are used to map pseudo-physical/physical addresses seen by the guest VM to the underlying memory of the machine. Since there is a vulnerability in Xen PV page tables that allows updates to be made to pre-existing page table entries, the malicious PV guests can access the page directory with an updated write privilege to execute the VM escape, breaking out of DomU to control Dom 0.

Both types of attacks were launched on the PV module configured in Qubes 3.1 with Xen 4.6 [22]. As illustrated in Figure 3, the attacker impersonating the PV guest root user could execute a command, "qvm-run victim firefox," that can only be executed by Dom0 to open the victim PV guest's Firefox web browser. Both attacks allowed the PV guest users to gain the control of Dom0.


[victim] Fedora Project - Start Page - Mozilla Firefox
Fedora Project - Start Pa...
https://start.fedoraproject.org
Search
fedora
A Red Hat-Sponsored Community Project
Fedora Documentation
Help for Fedora Users
Get Fedora
[attacker] root@attacker:/home/user/privesc_poc
File
Edit
View
Search
Terminal
Help
Qubes VM Manager
Fedora Project - Start Page - Mozilla Firefo
(no subject) - changweil@gmail.com - Gm
04:39 PM

Latest news from Fedora Magazine:

JUL 09 Boost your typing with emoji in Fedora 28 Workstation 4

Fedora 28 Workstation ships with a feature that allows you to quickly search, select and input emoji using your keyboard. Emoji, cute ideograms that are part of Unicode, are used fairly widely in messaging and especially on mobile devices. You may have heard the idiom “A picture is worth a thousand words.” This is exactly...

```
module.c                                           user_sh
ellcode.asm
module.o                                           user_sh
ellcode.bin
modules.order                                      user_sh
ellcode.bin.h
[root@attacker privesc_poc]# make
nasm -f elf64 -o /native.o native.asm
[root@attacker privesc_poc]# ./attack 'qvm-run victim
 firefox'
press enter to continue
```

Figure 3: CVE-2017-7228 and CVE-2016-6258 Attacks

4.2 Identifying Evidence Coverage for Forensic Analysis

The subsection describes the third step of our methodology that seeks to determine the gaps in evidence data that is required for fully detecting and reconstructing those attacks and to identify techniques required to gather missing evidence during subsequent attack runs. Towards this goal, the analysis of existing evidence was conducted as follows:

It has been observed that both attacks used vulnerabilities related to hypercalls and soft MMU in Xen in addition to using Xen’s device activity logs. The affected processes’ runtime syscalls were therefore logged to perform a forensic analysis. As an example, Appendix C illustrates the syscalls obtained by using the “strace” Linux command on the running “attack” program of CVE-2017-7228. Analysis of the device activity logs and runtime syscalls showed the relevant evidence originated from the syscalls captured from the attackers’ VMs. Despite the noise among syscalls that can be found in most programs, other syscalls revealed that the attack program injected a loadable kernel module into the kernel space which exploited the vulnerability to control the Dom0. This then opened the Firefox browser in the victim’s guest VM.

Evidence acquisition plays an important role in forensic analysis by determining and reconstructing attacks. As presented in a previous work which illustrated the use of a layered graphical framework to reconstruct attack scenarios [24], relevant evidence was identified and collected to reconstruct the corresponding attack path(s) representing the attack scenarios. During this process, an attack path with missing attack steps led to the collection of additional supporting evidence. An analysis of the syscalls captured for two sample attacks revealed that while the syscalls obtained using "strace" Linux command were useful for forensic analysis, they lacked attack details and had the following deficiencies: (1) the syscalls did not provide details of how features of the loadable kernel module used Xen's memory management to launch the attack; and (2) the syscalls were collected from the attacker's guest VM, which could easily be tampered with or removed by the attacker. The VM introspection technique and corresponding memory analysis tools are therefore recommended to obtain more supporting and admissible evidence from the run-time memory.

4.3 Use of Virtual Machine Introspection (VMI) for Forensics

The VMI is a process that allows for the external viewing of the state of a VM, either from a privilege VM or VMM itself. The state information includes CPU state (e.g., registers), all memory, and all I/O device states such as the contents of storage devices or register states of I/O controllers. Leveraging this capability, VMI-based applications can be built to perform forensic analysis in the following ways:

1. The VMI-based application can capture the entire memory and I/O state of a VM that is suspected of being compromised or attacked by taking a checkpoint (snapshot). The captured state of the running VM under observation can be compared to either: (a) a suspended VM in a known good state or (b) the original VM image from which the running VM was instantiated. [26].
2. A VMI-based application can be built to perform execution path analysis on the monitored VM. This is achieved by tracing—analyzing the sequence of VM activities and the corresponding complete VM state (e.g., memory map, IO access). This aids in the construction of a detailed attack graph with the VM state as nodes and the VM activities as edges, thereby tracing the path through which the current compromised state was reached [27]. This approach addresses deficiencies in performing forensic analysis that simply uses the system calls from the compromised VMs as follows:
    - There is the possibility that syscalls/hypercalls from the compromised VM could be tampered with or entirely removed by the attacker. In this approach, the sequence of VM states and VM activities are captured from outside the compromised VM, thus eliminating this possibility.
    - All variables that characterize a VM state and a VM activity are captured, helping to reconstruct the attack details based on memory access information with the ability to detect even malicious attacks, such as code and data modification.

Though VMI addresses deficiencies in forensic analysis that simply uses the system calls from the compromised VM, VMI tools must reconstruct the operational semantics of the guest operating system based on low-level sources such as physical memory and CPU registers [29]. Because LibVMI [30] provides VMI function on Xen and KVM,and bridges the semantic gap by reconstructing high-level state information from low-level physical memory data, we used LibVMI as the introspection tool to capture evidence from our two sample attacks. In order to use LibVMI on the two attacks, we installed Xen 4.6 in Debian 8 with the privileged Dom0 and both PV guests in DomU configured as Kernel 3.10.100 and Ubuntu 16.04.5, respectively. By running current LibVMI (release 0.12) installed on Dom0, we captured all running processes and injected Linux modules of the guest attack VM. We illustrated the processes captured during the attack time of CVE-2017-7228 in Appendix D, in which the two command lines following "root@debian" show the two programs vmi-process-list and vmi-module-list were executed to capture the running processes and modules of the attacker's VM, pv-attacker in our experiment. Lines between "[0]" and "[704]" are the captured processes (each line is composed of the number of the process, the name of the process and the kernel task list's address where the process name was retrieved); and lines following the command "vmi-module-list pv-attacker" are the captured modules (each line shows the module name). By comparing the captured processes and modules during the attack time with those from some other time (e.g., during normal execution) , it was easy to identify the attack process (named as attack in Line [704]) and the injected attack module (named as test as the first listed module. Its extension,.ko, is omitted by the program).

While an introspection tool such as LibVMI provides an effective way to detect the hypervisor attacks, it has limitations. First, to perform consistent memory access, LibVMI pauses and resumes the guest VM (e.g., our experiment showed that the attacker's VM was paused for about 0.035 756 s and 0.036 173 s for capturing the running process and injected modules, respectively). Second, because the VMI tool is only effective during the attack time, an attacker can easily utilize an in-VM timing mechanism, such as "kprobes" (the tracing framework built into kernel), to evade a passive VMI system [31]. Third, storing the captured snapshots of the guest VMs for forensic analysis often requires a large amount of storage space. Our current work addresses constructing the detailed attack path by analyzing the attacker's VM snapshots and improving upon the timing and memory issues related to using introspection.

5 Summary and Benefits

The vulnerabilities in two open source products, Xen and KVM, were analyzed and classified based on the hypervisor functionality where they exist, attack type and attack source. The analysis showed that most attacks on the two hypervisors were caused by vulnerabilities that existed in the soft MMU and I/O and Networking functionalities; the two most common hypervisor attack types were DoS and privilege escalation attacks; and that most attackers were guest OS users. Based on this information, two sample attacks were launched for forensic analysis with the log capture of system call data. The collected data on the sample attacks showed that evidence that is critically required for fully detecting and reconstructing those attacks was the runtime memory access information, but this information was missing from the log of the current run. VMI was identified as the requisite technique to gather this needed evidence and was incorporated in the subsequent attack run.

The intended benefit of the methodology is to enable all stakeholders (cloud providers and customers) to gain a better understanding of recent hypervisor vulnerabilities and attack trends, identify forensic information needed to reveal the presence of such attacks, and develop guidance on taking proactive steps to detect and prevent those attacks in their operating environments.

Appendix A—Xen and KVM Vulnerabilities

Table 5: The 83 Xen Vulnerability Entries in NIST-NVD (2016-2017)

No.	Xen-CVE-Entry	Functionality	No.	Xen-CVE- Entry	Functionality
1	CVE-2017-15588	Hypercall	43	CVE-2016-9381	MMU
2	CVE-2017-7228	Hypercall	44	CVE-2016-9383	MMU
3	CVE-2017-8903	hypercall	45	CVE-2016-9384	MMU
4	CVE-2016-3961	I/O and networking	46	CVE-2016-9385	MMU
5	CVE-2016-5403	I/O and Networking	47	CVE-2016-9386	MMU
6	CVE-2016-9637	I/O and networking	48	CVE-2016-9932	MMU
7	CVE-2016-9815	I/O and networking	49	CVE-2017-10912	MMU
8	CVE-2016-9816	I/O and networking	50	CVE-2017-10915	MMU
9	CVE-2016-9817	I/O and Networking	51	CVE-2017-10918	MMU
10	CVE-2016-9818	I/O and Networking	52	CVE-2017-14316	MMU
11	CVE-2017-10911	I/O and networking	53	CVE-2017-14431	MMU
12	CVE-2017-10913	I/O and networking	54	CVE-2017-15591	MMU
13	CVE-2017-10914	I/O and networking	55	CVE-2017-15592	MMU
14	CVE-2017-10920	I/O and networking	56	CVE-2017-15593	MMU
15	CVE-2017-10921	I/o and networking	57	CVE-2017-15595	MMU
16	CVE-2017-10922	I/O and networking	58	CVE-2017-15596	MMU
17	CVE-2017-12134	I/O and networking	59	CVE-2017-17044	MMU
18	CVE-2017-12135	I/O and networking	60	CVE-2017-17045	MMU
19	CVE-2017-12136	I/O and networking	61	CVE-2017-17046	MMU
20	CVE-2017-12137	I/O and Networking	62	CVE-2017-17563	MMU

No.	Xen-CVE-Entry	Functionality	No.	Xen-CVE- Entry	Functionality
21	CVE-2017-12855	I/O and networking	63	CVE-2017-17564	MMU
22	CVE-2017-14318	I/O and networking	64	CVE-2017-17565	MMU
23	CVE-2017-14319	I/O and networking	65	CVE-2017-17566	MMU
24	CVE-2017-15589	I/O and networking	66	CVE-2017-8905	MMU
25	CVE-2017-15597	I/O and networking	67	CVE-2016-7093	MMU
26	CVE-2017-7995	I/O and networking	68	CVE-2016-9382	MMU
27	CVE-2017-8904	I/O and networking	69	CVE-2016-3710	vCPU
28	CVE-2017-15594	Interrupt/Timer	70	CVE-2016-3712	vCPU
29	CVE-2016-7154	Interrupt/Timer	71	CVE-2016-6259	vCPU
30	CVE-2016-9377	Interrupt/Timer	72	CVE-2017-10916	vCPU
31	CVE-2016-9378	Interrupt/Timer	73	CVE-2017-10923	vCPU
32	CVE-2017-10917	Interrupt/Timer	74	CVE-2016-7777	vCPU
33	CVE-2017-10919	Interrupt/Timer	75	CVE-2016-10025	VM Exit
34	CVE-2017-15590	Interrupt/Timer	76	CVE-2016-4962	VM management
35	CVE-2016-10013	MMU	77	CVE-2016-4963	VM management
36	CVE-2016-10024	MMU	78	CVE-2016-9379	VM management
37	CVE-2016-3960	MMU	79	CVE-2016-9380	VM Management
38	CVE-2016-4480	MMU	80	CVE-2017-14317	VM Management
39	CVE-2016-5242	MMU	81	CVE-2017-5572	Vm Management
40	CVE-2016-6258	MMU	82	CVE-2017-5573	VM management
41	CVE-2016-7092	MMU	83	CVE-2016-5302	Remote management software

No.	Xen-CVE-Entry	Functionality	No.	Xen-CVE- Entry	Functionality
42	CVE-2016-7094	MMU			

Table 6: The 20 KVM Vulnerability Entries in NIST-NVD (2016-2017)

No.	KVM-CVE-Entry	Functionality	No.	KVM-CVE-Entry	Functionality
1	CVE-2016-0749	Adds-on	11	CVE-2017-12188	MMU
2	CVE-2016-5412	Hypercall	12	CVE-2016-8630	MMU
3	CVE-2017-15306	I/O and Networking	13	CVE-2017-2583	MMU
4	CVE-2016-10150	I/O and Networking	14	CVE-2016-9756	MMU
5	CVE-2016-3713	I/O and Networking	15	CVE-2017-12154	vCPU
6	CVE-2017-17741	I/O and Networking	16	CVE-2016-9777	vCPU
7	CVE-2016-4020	Interrupt/Timer	17	CVE-2017-2584	vCPU
8	CVE-2017-1000252	Interrupt/Timer	18	CVE-2017-12168	vCPU
9	CVE-2016-4440	Interrupt/Timer	19	CVE-2017-2596	VM Exit
10	CVE-2016-9588	MMU	20	CVE-2017-8106	VM Exit

Appendix B—Description of Hypervisor Functionality

Virtual CPUs (vCPU): A vCPU, also known as a virtual processor, abstracts a portion or share of a physical CPU that is assigned to a virtual machine (VM). The hypervisor uses a portion of the physical CPU cycle and allocates it to a vCPU assigned to a VM. The hypervisor schedules vCPU tasks to the physical CPUs.

Virtual Symmetric Multiprocessing (VSMP): VSMP is a method of symmetric multiprocessing (SMP), which enables multiple vCPU belonging to the same VM to be scheduled to a physical CPU that has at least two logical processors.

Soft Memory Management Unit (Soft MMU): The Memory Management Unit (MMU) is the hardware responsible for managing memory by translating the virtual addresses manipulated by the software into physical addresses. In an OS running on bare metal, the MMU translates the virtual addresses manipulated by the software into physical addresses. The mappings from virtual to physical addresses are kept in page tables (PT) and managed by the OS. In a virtualized environment, the hypervisor emulates the MMU (therefore called the soft MMU) for the guest OSs. This is done by mapping what the guest OS sees as physical memory (often called pseudo-physical/physical address in Xen) to the underlying memory of the machine (called machine addresses in Xen). The mapping table from the physical address to machine address (P2M) is typically maintained in the hypervisor and hidden from the guest OS by using techniques such as a shadow page table (SPT) for each guest VM[16, 17]. When in SPT mode, the guest OS PT is not performing a mapping from virtual-to-machine, but a virtual-to-physical mapping. The Xen paravirtualized MMU model requires that the guest OS be directly aware of mapping between (pseudo) physical and machine addresses (the P2M table). Additionally, in order to read page table entries that contain machine addresses and convert them back into (pseudo) physical addresses, a translation from machine to (pseudo) physical addresses provided by the M2P table is required in Xen paravirtualized MMU model [17].

I/O and Networking: There are three common approaches that provide I/O services to guest VMs. Using the Xen I/O structures illustrated in Figure 4 as an example, these common approaches include:

> (1) the hypervisor emulates a known I/O device in a fully virtualized system, and the guests use an unmodified driver (called a native driver) to interact with it (illustrated as "Native Driver 1" in DomU to "Device Model" in Dom0 in Figure 4);
>
> (2) a paravirtual driver (known as a front-end driver) in a paravirtualized system is installed in the modified guest OS in DomU, which uses shared-memory—asynchronous buffer-descriptor rings—to communicate with the back-end I/O driver in the hypervisor (illustrated as "Front-end Driver" in DomU to "Back-end Driver" to Dom0 in Figure 4);
>
> (3) the host assigns a device (known as a pass-through device) directly to the guest VM (illustrated as "Native Driver 2" in DomU to "Pass-through Device" in Figure 4).

To reduce I/O virtualization overhead, improve virtual machine performance, and provide I/O services to guest VMs, scalable self-virtualizing I/O devices that allow direct access interface to multiple VMs are also used. However, the two approaches do not virtualize the I/O since they include direct access, and self-virtualized I/O devices allow the device driver within a guest OS to interact with the hardware directly. Furthermore, they scale poorly due to challenges, performance, and cost [22].

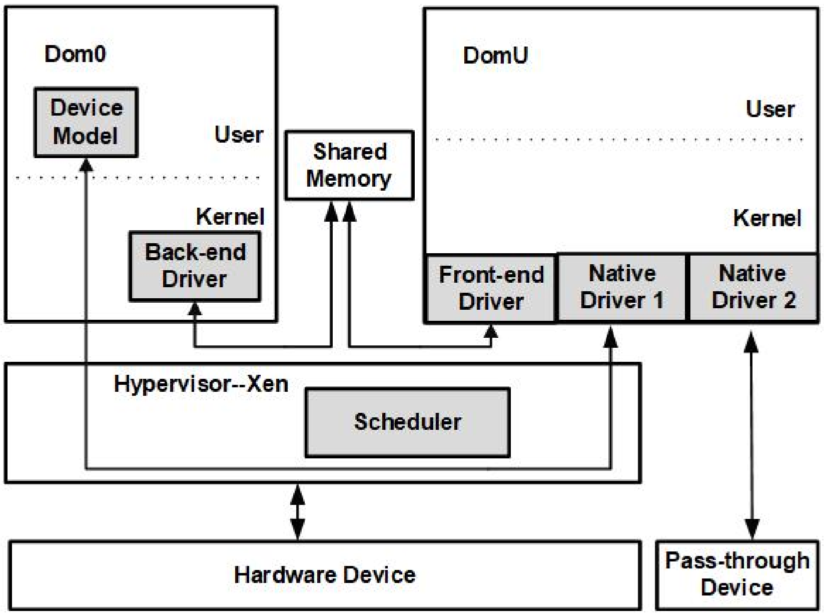


Figure 4: Xen I/O structures

In paravirtualized Xen systems, the front-end and back-end drivers communicate with each other using two producer-consumer ring buffers (standard lockless shared memory data structures built on grant tables and event channels), where one is used for packet reception and the other is used for packet transmission. Though hypervisors enforce isolation across VMs residing within a single physical machine, the grant mechanism provides inter-domain communications in Xen, allowing shared-memory communications between unprivileged domains by using grant tables [16]. Grant tables are used to protect the I/O buffer in a guest domain's memory and share the I/O buffer with Dom0 properly, which underpin the split device drivers for block and network I/O. Each domain has its own grant table that allows the domain to inform Xen with the kind of permissions other domains have on their pages. KVM typically uses Virtio, a virtualization standard for network and disk drivers, which is architecturally similar to Xen paravirtualized device drivers which are composed of front-end drivers and back-end drivers.

Interrupt/Timer: Hypervisors should be able to virtualize and manage interrupts/timers [18], the interrupt/timer controller of the guest OS, and the guest OS’s access to the controller. The interrupt/timer mechanism in a hypervisor includes a programmable interval timer (PIT), the advanced programmable interrupt controller (APIC), and the interrupt request (IRQ) mechanisms.

Hypercall: Hypercalls are similar to system calls (syscalls) that provide user-space applications with kernel-level operations. They are performed using the syscall instruction with up to six arguments passed in registers. A hypercall layer is commonly available and allows guest OSs to make requests of the host OS. Domains will use hypercalls to request privileged operations such as updating page tables from the hypervisors. Thus, an attacker can use hypercalls to attack the hypervisor from a guest VM.

VMExit: According to Belay at el. [19], the mode change from Virtual Machine Extension (VMX) root mode to VMX non-root mode is called VMEntry, and the mode change from VMX non-root mode to VMX root mode is called VMExit. VM exits are a response to some instructions and events (e.g., page fault) from guest VMs and are the main cause of performance degradation in a virtualized system. These events could include external interrupts, accesses to control registers, task switches, and I/O operation instructions (e.g., INB, OUTB).

VM management functionality: Hypervisors support basic VM management functionalities, including starting, pausing, or stopping VMs. These tasks are managed in Xen Dom0 and KVM's libvirt driver.

Remote Management Software: Remote management software is employed as a user-friendly interface that remotely manages the hypervisor through the network. With an intuitive user interface that visualizes the status of a system, the remote management software allows administrators to tweak or manage the virtualized environment.

Add-ons: The add-ons of hypervisors use modular designs to add extended functions. By leveraging the interaction between the add-ons and hypervisors, an attacker can cause a host to crash (a DoS attack) or even compromise the host.

Appendix C—The Syscalls Intercepted from the Attacking Program

The syscalls in this appendix were obtained by employing the Linux command "strace" on the running attack program using the vulnerability CVE-2017-7228 (the attack program is named "attack"). These syscalls show: (1) the attacker executed the attack program with arguments aimed at the victim guest VM (Line 1); (2) the attack program and required Linux libraries have been loaded to the memory for the program execution (Line 2 to Line 16); (3) the memory pages of the attack program have been protected from access by other processes (Line 17 to Line 23); and (4) the attack program injected a loadable Linux module named "test.ko" to the kernel space to exploit the vulnerability (Line 24 to Line 31).

```
1.  execve("./attack", ["./attack", "qvm-run victim firefox"], [/* 30 vars */]) = 0
2.  brk(NULL)                          = 0x8cd000
3.  mmap(NULL, 4096, PROT_READ|PROT_WRITE,
    MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fa3a3022000
4.  access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
5.  open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
6.  fstat(3, {st_mode=S_IFREG|0644, st_size=74105, ...}) = 0
7.  mmap(NULL, 74105, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fa3a300f000
8.  close(3)                           = 0
9.  open("/lib64/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
10. read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\240\6\2\0\0\0\0\0"..., 832) = 832
11. fstat(3, {st_mode=S_IFREG|0755, st_size=2104216, ...}) = 0
12. mmap(NULL, 3934688, PROT_READ|PROT_EXEC,
    MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fa3a2a42000
13. mprotect(0x7fa3a2bf9000, 2097152, PROT_NONE) = 0
14. mmap(0x7fa3a2df9000, 24576, PROT_READ|PROT_WRITE,
    MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b7000) = 0x7fa3a2df9000
15. mmap(0x7fa3a2dff000, 14816, PROT_READ|PROT_WRITE,
    MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fa3a2dff000
16. close(3)                           = 0
17. mmap(NULL, 4096, PROT_READ|PROT_WRITE,
    MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fa3a300e000
18. mmap(NULL, 4096, PROT_READ|PROT_WRITE,
    MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fa3a300d000
19. mmap(NULL, 4096, PROT_READ|PROT_WRITE,
    MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fa3a300c000
20. arch_prctl(ARCH_SET_FS, 0x7fa3a300d700) = 0
21. mprotect(0x7fa3a2df9000, 16384, PROT_READ) = 0
22. mprotect(0x600000, 4096, PROT_READ)     = 0
23. mprotect(0x7fa3a3023000, 4096, PROT_READ) = 0
24. munmap(0x7fa3a300f000, 74105)           = 0
25. open("test.ko", O_RDONLY)               = 3
26. finit_module(3, "user_shellcmd_addr=1407334317317"..., 0) = 0
27. fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 0), ...}) = 0
28. mmap(NULL, 4096, PROT_READ|PROT_WRITE,
    MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fa3a3021000
```

```
29. mmap(0x600000000000, 4096, PROT_READ|PROT_WRITE,
    MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS|MAP_LOCKED, -1, 0) =
    0x600000000000
30. delete_module("test", O_NONBLOCK)      = 0
31. exit_group(0)                          = ?
```

Appendix D—Forensic Data Obtained by Using LibVMI

The running processes obtained from the attacker's VM:

```
root@debian:/home/guest/src/libvmi/libvmi# ./examples/vmi-process-list pv-attacker
Process listing for VM pv-attacker (id=2)
[    0] swapper/0 (struct addr:ffffffff81e13500)
[    1] systemd (struct addr:ffff88007c460000)
[    2] kthreadd (struct addr:ffff88007c460e00)
[    3] ksoftirqd/0 (struct addr:ffff88007c461c00)
[    4] kworker/0:0 (struct addr:ffff88007c462a00)
[    5] kworker/0:0H (struct addr:ffff88007c463800)
[    6] kworker/u2:0 (struct addr:ffff88007c464600)
[    7] rcu_sched (struct addr:ffff88007c465400)
[    8] rcu_bh (struct addr:ffff88007c466200)
[    9] migration/0 (struct addr:ffff88007c467000)
[   10] watchdog/0 (struct addr:ffff88007c4c8000)
[   11] kdevtmpfs (struct addr:ffff88007c4c8e00)
[   12] netns (struct addr:ffff88007c4c9c00)
[   13] perf (struct addr:ffff88007c4caa00)
[   14] xenwatch (struct addr:ffff88007c4cb800)
[   15] xenbus (struct addr:ffff88007c4cc600)
[   16] khungtaskd (struct addr:ffff88007c4cd400)
[   17] writeback (struct addr:ffff88007c4ce200)
[   18] ksmd (struct addr:ffff88007c4cf000)
[   19] crypto (struct addr:ffff88007c568000)
[   20] kintegrityd (struct addr:ffff88007c568e00)
[   21] bioset (struct addr:ffff88007c569c00)
[   22] kblockd (struct addr:ffff88007c56aa00)
[   23] ata_sff (struct addr:ffff88007c56b800)
[   24] md (struct addr:ffff88007c56c600)
[   25] devfreq_wq (struct addr:ffff88007c56d400)
[   26] kworker/0:1 (struct addr:ffff88007c56e200)
[   27] kworker/u2:1 (struct addr:ffff88007c56f000)
[   29] kswapd0 (struct addr:ffff880076060e00)
[   30] vmstat (struct addr:ffff880076061c00)
[   31] fsnotify_mark (struct addr:ffff880076062a00)
[   32] ecryptfs-kthrea (struct addr:ffff880076063800)
[   48] kthrotld (struct addr:ffff880076131c00)
[   50] khvcd (struct addr:ffff880076133800)
[   51] bioset (struct addr:ffff880076134600)
[   52] bioset (struct addr:ffff880076135400)
[   53] bioset (struct addr:ffff880076136200)
[   54] bioset (struct addr:ffff880076137000)
[   55] bioset (struct addr:ffff880076238000)
[   56] bioset (struct addr:ffff880076238e00)
[   57] bioset (struct addr:ffff880076239c00)
[   58] bioset (struct addr:ffff88007623aa00)
[   62] ipv6_addrconf (struct addr:ffff88007623e200)
[   64] bioset (struct addr:ffff880076132a00)
[   76] deferwq (struct addr:ffff880076100000)
[   77] charger_manager (struct addr:ffff880076100e00)
[  145] kworker/0:1H (struct addr:ffff880004340e00)
[  147] jbd2/xvda1-8 (struct addr:ffff880004340000)
[  148] ext4-rsv-conver (struct addr:ffff880004346200)
[  177] kworker/0:2 (struct addr:ffff880076103800)
[  181] kworker/0:3 (struct addr:ffff880076065400)
[  184] kworker/0:4 (struct addr:ffff880076064600)
[  187] kworker/0:5 (struct addr:ffff880004347000)
[  189] kworker/0:6 (struct addr:ffff880004345400)
[  192] kworker/0:7 (struct addr:ffff88007623f000)
[  194] kworker/0:8 (struct addr:ffff88007623c600)
[  195] systemd-journal (struct addr:ffff88007623b800)
[  196] kauditd (struct addr:ffff880076130e00)
[  198] kworker/0:9 (struct addr:ffff880078820000)
[  201] kworker/0:10 (struct addr:ffff880078823800)
[  204] kworker/0:11 (struct addr:ffff880078825400)
[  206] kworker/0:12 (struct addr:ffff880078827000)
[  207] kworker/0:13 (struct addr:ffff880076105400)
[  234] systemd-udevd (struct addr:ffff880076106200)
[  383] systemd-timesyn (struct addr:ffff88007bd7aa00)
[  516] dhclient (struct addr:ffff880077b78e00)
[  560] cron (struct addr:ffff880077b7d400)
[  562] dbus-daemon (struct addr:ffff880076107000)
[  574] accounts-daemon (struct addr:ffff880004342a00)
[  577] systemd-logind (struct addr:ffff880078821c00)
[  579] rsyslogd (struct addr:ffff880078820e00)
[  615] sshd (struct addr:ffff880003c88000)
[  629] login (struct addr:ffff880003c8b800)
[  630] agetty (struct addr:ffff880003c8e200)
[  669] systemd (struct addr:ffff880076060000)
[  674] (sd-pam) (struct addr:ffff880076104600)
[  677] bash (struct addr:ffff880003c8aa00)
[  703] sudo (struct addr:ffff880004341c00)
[  704] attack (struct addr:ffff880004343800)
```

The obtained injected modules from the attacker's VM:

```
root@debian:/home/guest/src/libvmi/libvmi# ./examples/vmi-module-list pv-attacker
test
intel_rapl
x86_pkg_temp_thermal
coretemp
crct10dif_pclmul
```

```
crc32_pclmul
ghash_clmulni_intel
aesni_intel
aes_x86_64
lrw
gf128mul
glue_helper
ablk_helper
cryptd
autofs4
```

Appendix E—References

[1] Uhlig R, Neiger G, Rodgers D, Santoni AL, Martins FCM, Anderson AV, Bennett SM, Kagi A, Leung FH, Smith L (2005) Intel virtualization technology. *Computer* 38(5):48-56. https://doi.org/10.1109/MC.2005.163

[2] Mell P, Grance T (2010) The NIST definition of cloud computing. *Communications of the ACM* 53(6):50-54.

[3] Goldberg RP (1974) Survey of virtual machine research. *Computer* 7(6):34-45. https://doi.org/10.1109/MC.1974.6323581

[4] Perez-Botero D, Szefer J, Lee RB (2013) Characterizing hypervisor vulnerabilities in cloud computing servers. *Proceedings of the 2013 International Workshop on Security in Cloud Computing,* (ACM, Hangzhou, China), pp 3-10. https://doi.org/10.1145/2484402.2484406

[5] Thongthua A, Ngamsuriyaroj S (2016) Assessment of hypervisor vulnerabilities. *Proceedings of the 2016 International Conference Cloud Computing Research and Innovations* (ICCCRI), (IEEE, Singapore, Singapore), pp 71-77. https://doi.org/10.1109/ICCCRI.2016.19

[6] Szefer J, Keller E,Lee RB, Rexford R (2011) Eliminating the hypervisor attack surface for a more secure cloud. *Proceedings of the 18th ACM Conference on Computer and Communications Security,* (ACM, Chicago, IL), pp 401-412. https://doi.org/10.1145/2046707.2046754

[7] Graziano M, Lanzi A, Balzarotti D (2013) Hypervisor memory forensics. *Proceedings of the International Workshop on Recent Advances in Intrusion Detection*, (Springer, Rodney Bay, St. Lucia), pp 21-40. https://doi.org/10.1007/978-3-642-41284-4_2

[8] Joshi LM, Kumar M, Bharti R (2015) Understanding threats in hypervisor, its forensics mechanism and its research challenges. *International Journal of Computer Applications* 119(1):1-5. https://doi.org/10.5120/21028-2755

[9] Popek GJ, Goldberg RP (1974) Formal requirements for virtualizable third generation architectures. *Communications of the ACM* 17(7):412-421. https://doi.org/10.1145/361011.361073

[10] Pariseau B (2018) *KVM reignites Type 1 vs Type 2 hypervisor debate*. Available at https://searchservervirtualization.techtarget.com/news/2240034817/KVM-reignites-Type-1-vs-Type-2-hypervisor-debate

[11] Xen Project (2019) *Xen project software overview.* Available at https://wiki.xen.org/wiki/Xen_Project_Software_Overview

[12] KVM contributors (2019) *Kernel Virtual Machine* [Main page]. Available at https://www.linux-kvm.org/page/Main_Page

[13] Shi J, Yang Y, Tang C (2016) Hardware assisted hypervisor introspection. *SpringerPlus*, 5:647. https://doi.org/10.1186/s40064-016-2257-7

[14] National Institute of Standards and Technology (2019) *National Vulnerability Database*. Available at https://nvd.nist.gov

[15] QEMU contributors (2019) *QEMU--the FAST! processor emulator*. Available at https://www.qemu.org

[16] Kloster JF, Kristensen J, Mejlholm A (2006) Efficient memory sharing in the Xen virtual machine monitor. (Aalborg University). Available at http://mejlholm.org/uni/pdfs/dat5.pdf

[17] Xen Project (2019) *X86 Paravirtualised Memory Management*. Available at https://wiki.xen.org/wiki/X86_Paravirtualised_Memory_Management

[18] Song Y, Wang H, Soyata T (2015) Hardware and software aspects of VM-based mobile-cloud offloading. *Enabling Real-Time Mobile Cloud Computing through Emerging Technologies,* ed Soyata T (Hershey, PA, Information Science Reference), Chapter 8, 1st Ed., pp 247-271. https://doi.org/10.4018/978-1-4666-8662-5.ch008

[19] Belay A, Bittau, A, Mashtizadeh J, Terei D, Mazières D, Kozyrakis C (2012). Dune: safe user-level access to privileged CPU features. Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation, (USENIX Association, Hollywood, CA), pp 335-348. https://www.usenix.org/node/170864

[20] Horn J (2017) *Paravirtualization: exploiting the Xen hypervisor*. Available at https://googleprojectzero.blogspot.com/2017/04/pandavirtualization-exploiting-xen.html

[21] Boutoille J, Campana G (2016) *Xen exploitation part 3: XSA-182, Qubes escape*. Available at https://blog.quarkslab.com/xen-exploitation-part-3-xsa-182-qubes-escape.html

[22] Satran J, Shalev L, Ben-Yehuda M, Machulsky Z (2008) Scalable I/O-a well-architected way to do scalable, secure and virtualized I/O. *First Workshop on I/O Virtualization (WIOV'08)*, (USENIX Association, San Diego, CA), pp 1-6. Available at https://www.usenix.org/legacy/events/wiov08/tech/full_papers/satran/satran.pdf

[23] Lowe SD (2015) 2015 state of hyperconverged infrastructure market report. (ActualTech Media, Bluffton, SC). Available at https://www.actualtechmedia.com/wp-content/uploads/2015/05/2015-State-of-Hyperconverged-Infrastructure-Market-Report.pdf

[24] Liu C, Singhal A, Wijesekera D (2017) A layered graphical model for mission attack impact analysis. *IEEE Conference on Communications and Network Security (CNS)*, (IEEE, Las Vegas, NV), pp 602-609. https://doi.org/10.1109/CNS.2017.8228706

[25] Fayyad-Kazan H, Perneel L, Timmerman M (2013) Full and para-virtualization with Xen: a performance comparison. *Journal of Emerging Trends in Computing and Information Sciences* 4(9):719-727. Available at http://www.cisjournal.org/journalofcomputing/archive/vol4no9/vol4no9_9.pdf

[26] Garfinkel T, Rosenblum M (2003) A Virtual Machine Introspection Based Architecture for Intrusion Detection. *Proceedings of Network and Distributed Systems Security Symposium*, (Internet Society, San Diego, CA), pp 1-16. Available at https://www.ndss-symposium.org/wp-content/uploads/2017/09/A-Virtual-Machine-Introspection-Based-Architecture-for-Intrusion-Detection-Tal-Garfinkel.pdf

[27] Moser A, Kruegel C, Kirda E (2007) Exploring multiple execution paths for malware analysis. Proceedings of the IEEE Symposium on Security and Privacy (S&P 2007), (IEEE, Berkeley, CA), pp 231-245. https://doi.org/10.1109/SP.2007.17

[28] Whalen E (2013) *HVM vs. Paravirtualized*. (Performance Tuning Corporation, Austin, TX). Available at https://www.perftuning.com/blog/hvm-vs-paravirtualized

[29] Dolan-Gavitt B, Payne B, Lee W (2011) Leveraging forensic tools for virtual machine introspection. (Georgia Institute of Technology, Atlanta, GA), Technical Report, GT-CS-11-05. http://hdl.handle.net/1853/38424

[30] LibVMI contributors (2015) *LibVMI-Virtual Machine Introspection*. Available at http://libvmi.com

[31] Wang G, Zachary JE, Pham CM, Kalbarczyk ZT, Iyer RK (2015) Hypervisor Introspection: A Technique for Evading Passive Virtual Machine Monitoring. *9th USENIX Workshop on Offensive Technologies (WOOT '15)*, (USENIX Association, Washington, DC), pp 1-8. https://www.usenix.org/node/191959