Draft NISTIR 8221

A Methodology for Determining Forensic Data Requirements for Detecting Hypervisor Attacks

Ramaswamy Chandramouli
Anoop Singhal
Duminda Wijesekera
Changwei Liu

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

Draft NISTIR 8221

A Methodology for Determining Forensic Data Requirements for Detecting Hypervisor Attacks

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

September 2018

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 Internal Report 8221
27 pages (September 2018)

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.

Public comment period: *September 21, 2018* through *October 12, 2018*

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 key feature of data centers used for cloud computing services and enterprise computing that enables ubiquitous access to shared system resources. Server virtualization is typically performed by a hypervisor, which provides mechanisms to abstract hardware and system resources from an operating system. Hypervisors are large pieces of software with several thousand lines of code and are therefore known to have vulnerabilities. This document analyzes the recent vulnerabilities associated with two open-source hypervisors—Xen and KVM—as reported by the National Institute of Standards and Technology's (NIST) National Vulnerability Database (NVD), and develops a profile of those vulnerabilities in terms of hypervisor functionality, attack type, and attack source. Based on the predominant number of vulnerabilities in a hypervisor functionality (attack vector), two sample attacks using those attack vectors were launched to exploit those vulnerabilities, and the associated system calls were logged. The objective was to determine the evidence coverage for detecting and reconstructing those attacks and identify techniques required to gather missing evidence.

Keywords

cloud computing; forensic analysis; hypervisors; 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

Hypervisors provide the mechanism that both creates and runs multiple operating systems (also called guest virtual machines) on a single physical platform (a host) in cloud environments. The increasing popularity of cloud services and the complex nature of hypervisors, which are essentially large software modules, have led to malicious attackers exploiting hypervisor vulnerabilities in order to attack cloud services. To discover recent trends in hypervisor attacks and prevent future hypervisor exploitation, recent vulnerability reports associated with two popular open-source hypervisors in the NIST National Vulnerability Database (NIST-NVD), Xen and KVM, were analyzed and classified based on the hypervisor functionalities (attack vector), attack type and attack source.

Ten functionalities traditionally provided by hypervisors re considered for the classification of hypervisor vulnerabilities. These functionalities include: (1) virtual CPUs, (2) symmetric multiprocessing, (3) soft memory management unit, (4) interrupt and timer mechanisms, (5) I/O and networking, (6) paravirtualized I/O, (7) VM exits, (8) hypercalls, (9) VM management and remote management software, and (10) hypervisor Add-ons. Based on functionalities, the vulnerability profile reveals that most attacks were caused by vulnerabilities in the soft memory management unit and I/O and networking functionalities. It also reveals that two most common hypervisor attacks are denial-of-service (DoS) and privilege escalation attacks launched primarily by guest OS users. Using vulnerabilities related to the hypervisor functionality of the soft memory management unit, two sample attacks were launched to obtain the evidence needed to perform forensic analysis on hypervisor attacks in which corresponding system calls were captured. The objective was to determine the evidence coverage for detecting and reconstructing those attacks and identify techniques required to gather missing evidence. A close analysis of these system calls reveals that more evidence regarding the execution path for the attacks is found in the run-time memory.

The methodology outlined in this document can assist cloud providers in enhancing the security of their virtualized infrastructure, help cloud service customers discover recent hypervisor attack trends, identify information that reveals the presence of such attacks, and provide guidance on taking proactive steps to prevent those attacks in the operating environment.

Table of Contents

Executive Summary .... **iv**
1 Introduction **6**
2 Background and Related Work **7**
2.1 Hypervisors 7
2.1.1 Xen 7
2.1.2 KVM 8
2.2 Related Work 8
3 Deriving a Profile of Hypervisor Vulnerabilities **10**
3.1 The Vulnerabilities in the NIST-NVD 10
3.2 Associating Hypervisor Functionalities with Vulnerabilities 10
3.3 Deriving the Hypervisor Vulnerability Profile 12
4 Sample Attacks and Forensic Analysis **15**
4.1 The Two Sample Attacks 15
4.2 Identifying Evidence Coverage for Forensic Analysis 16
4.3 Use of Virtual Machine Introspection (VMI) for Forensics 17
5 Conclusions **18**
Appendix A— Description of Hypervisor Functionality **19**
Appendix B— The Syscalls Intercepted from the Attacking Program **22**
Appendix C— References **24**

1 Introduction

Most cloud services are provided in a virtualized environment. Since virtualization of all system resources—including processors, memory, and I/O devices—makes it possible to run multiple operating systems on a single physical platform (host), virtualization is a key feature of cloud computing 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 framework for conducting forensic analysis on popular hypervisors, such as 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 discover recent trends in hypervisor attacks, provide suggestions for mitigating hypervisor attack risks, and identify evidence of those attacks. The main contributions are as follows: (1) all recent hypervisor vulnerabilities of Xen and KVM (from years of 2016 and 2017) in NIST-NVD were analyzed and classified based on the hypervisor functionalities, the attack types, and the sources of attacks; (2) classifications of the recent Xen and KVM hypervisor vulnerabilities can provide suggestions for mitigating potential hypervisor attacks and enhancing the hypervisor resilience against known hypervisor vulnerabilities; (3) some sample attacks were simulated to show the methodology of determining the forensic data for detecting hypervisor attacks.

The rest of the publication is organized as follows. Section 2 presents the background of hypervisors and discusses related work. Section 3 lists typical hypervisor functionalities and shows analysis of the recent two-year hypervisor vulnerabilities listed in NIST-NVD. Section 4 describes the sample attacks and the forensic evidence used for reconstructing the sample attacks. Section 5 provides conclusions.

2 Background and Related Work

This section provides an outline of the architectures of the two open-source hypervisors and discusses related work in the area of cloud forensic analysis.

2.1 Hypervisors

Hypervisors are software and/or firmware modules that virtualize system resources such as CPU, memory, and devices. In [9], Popek and Goldberg classify hypervisors as Type 1 hypervisor and Type 2 hypervisor. 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 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 popular hypervisors: Microsoft Hyper-V, VMware VSphere/ESX, Citrix XenServer/Xen, and KVM. Since Microsoft Hyper-V and VMware VSphere/ESX are commercial products, this document and research focus on the vulnerabilities on two widely used open-source hypervisors, Xen and KVM. Their architectures are briefly discussed below.

2.1.1 Xen

Figure 1 shows the architecture of Xen. In this design, the Xen hypervisor manages three kinds of VMs including the control domain (also called Dom0) and guest domains (also called DomU) that support two different virtualization modes: Paravirtualization (PV) and Hardware-assisted Virtualization (HVM) [11]. 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 (Quick Emulator), a generic hardware emulator [15], for simulating PC hardware (e.g., CPU, BIOS, IDE, VGA, network cards, and USBs). Because of the use of simulation technologies, HVM VMs' performance is inferior to PV VMs. Xen 4.4 provides a new virtualization mode named 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, virtualized interrupts, and a boot. PVH guests require PVH-enabled guest OS [11].

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, KVM kernel module turns Linux kernel into a Type 1 bare-metal hypervisor, providing the power and functionality of even the most complex and powerful Type 1 hypervisors.

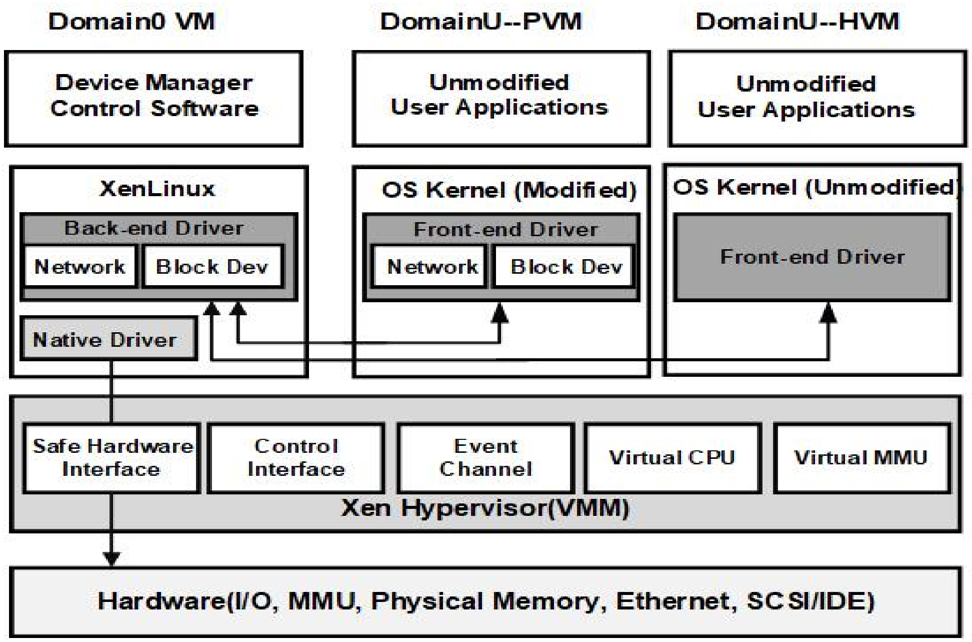

Figure 1: The Xen architecture

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 cannot be used to predict recent attack trends. To assess the weakness, severity scores, and attack impacts, Thongthua et al. assessed the vulnerabilities of widely used hypervisors, including VMware ESXi, Citrix XenServer, and KVM, using the NIST 800-115 security testing framework and performed some sample experiments [5]. 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].

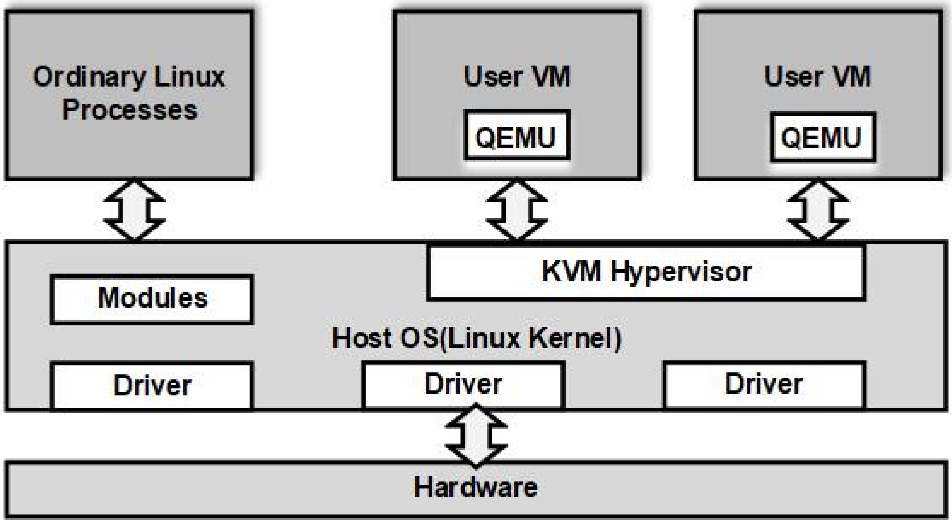

Figure 2: The KVM architecture

3 Deriving a Profile of Hypervisor Vulnerabilities

As a prelude to developing a methodology for determining forensic data requirements for detecting hypervisor attacks, it is necessary to derive a profile of recent hypervisor vulnerabilities in terms of the following classification criteria:

- Hypervisor Functionality where the vulnerability exists (attack vector)
- Attack Type (impact of the attack by exploiting the vulnerability)
- Attack Source (the component in the hypervisor platform from which the attack is launched)

The approach adopted for deriving the vulnerability profile involved obtaining all vulnerabilities (tagged with CVE numbers) in two open-source hypervisors (Xen and KVM) from the NIST-NVD for years 2016 and 2017. The hypervisor functionality (attack vector) was then associated with the attack type (impact) that resulted from exploiting each vulnerability and the attack source based on the description of vulnerabilities in that database. The total number of vulnerabilities for the two chosen open-source hypervisors in each of the three categories (attack vector, attack type, and attack source) thus provided a recent vulnerability profile for those hypervisor offerings.

A brief description of the information sources that were used and the steps adopted as part of the approach for deriving the vulnerability profile is given in sections 3.1, 3.2, and 3.3.

3.1 The Vulnerabilities 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]. A search of the NIST-NVD for the vulnerabilities posted during the years 2016 and 2017 revealed 83 Xen hypervisor vulnerabilities and 20 KVM hypervisor vulnerabilities. These vulnerabilities were then associated with the following:

- Hypervisor functionality where the vulnerability arises
- Potential attack type
- Attack source (i.e., the component/associated user from which the potential attack can be launched)

3.2 Associating Hypervisor Functionalities with Vulnerabilities

To better understand different hypervisor vulnerabilities, Perez-Botero et al. considered 11 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

Based on the common function provided by numbers four and five above, these were merged into a single functionality. (A detailed description of all these functionalities can be found in Appendix A). All reported Xen and KVM vulnerabilities during the years 2016 and 2017 were mapped to these hypervisor functionalities based on the approach in [4]. A brief description of a sample vulnerability associated with each functionality is given in Table 1 below:

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 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 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.

Hypervisor Functionality	Sample Vulnerability
VMExit	The exploit on VM Exit-handling code usually leads to a DoS attack. An example of 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 VMX root mode. This then 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	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.
Remote Management Software	NONE
Hypervisor Add-ons	CVE-2016-0749 is an example vulnerability of hypervisor add-ons. By leveraging the smartcard interaction in 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 Deriving the Hypervisor Vulnerability Profile

With the goal of deriving the hypervisor security vulnerability profile, 83 Xen and 20 KVM vulnerabilities listed in the NIST-NVD for the years 2016 and 2017 were analyzed and classified according to functionalities, attack types (impacts), and attack sources.

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

Number	Hypervisor Functionality	Xen	KVM
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%)

Number	Hypervisor Functionality	Xen	KVM
6	Hypercalls	3 (4%)	1 (5%)
7	VMExit	1 (1%)	2 (10%)
8	VM Management	8 (10%)	0 (0%)
9	Remote Management Software	0 (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 two functionalities of virtual symmetric multiprocessing and remote management software, 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, which 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	KVM
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. Table 3 reveals that the most common attack was DoS (44% for Xen and 63% for KVM), indicating that attacking cloud services' availability has been the most 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 three attacks occurs with less frequency than a DoS attack, they all result in more serious damage by allowing attackers to obtain sensitive user information or compromise 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), though other sources included cloud administrators, guest OS administrators, and remote users. 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	KVM
Administrator	2 (Management) (2%)	0 (0%)
Guest OS administrator	17 (including HVM and PV administrators) (20%)	1 (5%)
Guest OS user	63 (including 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

Since numerous vulnerabilities are related to Xen soft MMU functionality, this section will show two sample attacks, including those that exploit vulnerabilities CVE-2017-7228 and CVE-2016-6258, to demonstrate how the 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 manages three kinds of VMs, including the control domain (also called Dom0) and guest domains (also called DomU). These then support two different virtualization modes: Paravirtualization (PV) and Hardware-assisted Virtualization (HVM). The PV module has been widely utilized for its higher performance [25]. However, because the Xen PV model 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, allowing for VM escape (the process of breaking out of a guest VM and interacting with the hypervisor's host operating system) from DomU to Dom0, hypervisor host crash, and information leakage. With these attacks, the PV guest from "attacker" (the green terminal) could execute commands like "qvm-run victim firefox" to open a Firefox web-browser in "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 guests users to gain the control of Dom0.

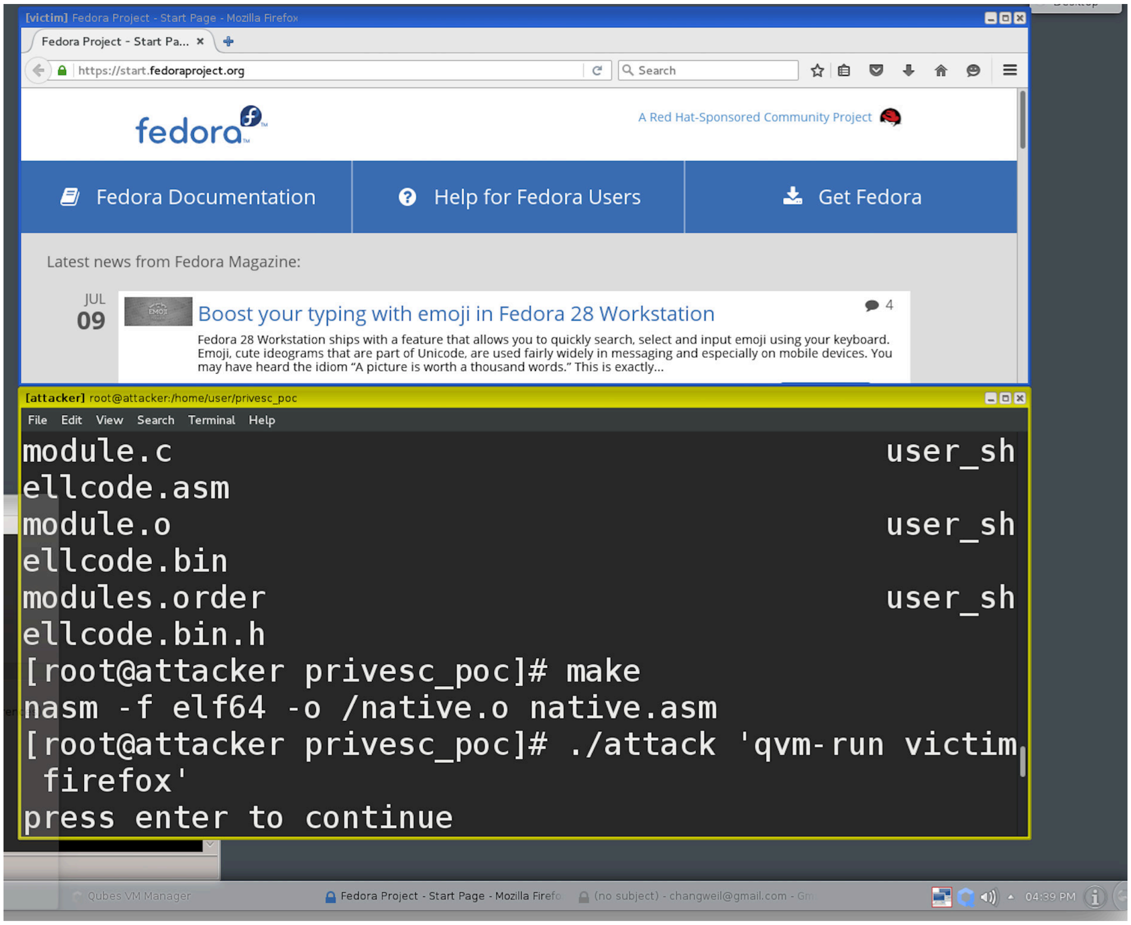

Figure 3: CVE-2017-7228 and CVE-2016-6258 Attacks

4.2 Identifying Evidence Coverage for Forensic Analysis

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 B 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 (taking a 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 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.

5 Conclusions

An analysis of all reported vulnerabilities on Xen and KVM in the last two years was conducted, and two sample attacks were launched to identify evidence for a forensic analysis. Data subsequently showed that most attacks on the two hypervisors were caused by vulnerabilities that existed in soft MMU and I/O and Networking functionalities. The two most common hypervisor attacks were DoS and privilege escalation attacks. Most attackers are guest OS users. The collected evidence on the sample attacks showed that most valuable evidence remains in the run-time system memory. Therefore, to obtain valuable evidence with guaranteed integrity, VM introspection technique and secure logging systems showing memory access should be implemented and used.

Appendix A—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 a shadow page table for each guest VM. Each shadow page table mapping translates virtual addresses of programs in a guest VM to guest (pseudo) physical addresses and is placed in the guest OS [16, 17]. 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].

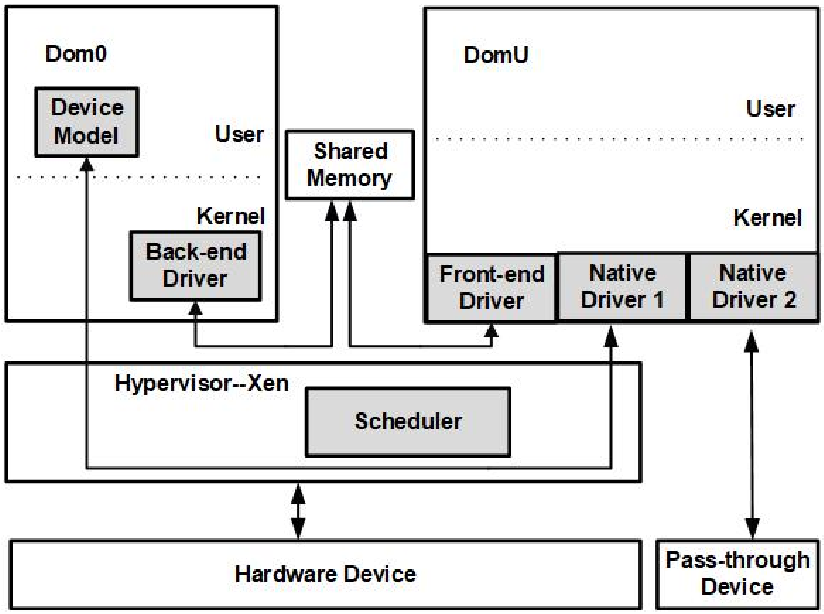

Figure 4: Xen I/O structures

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 [4].

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, triple faults, task switches, I/O operation instructions (e.g., INB, OUTB), and accesses to control registers.

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

Remote Management Software: Remote management software is employed as a user-friendly interface that connects directly to the hypervisor in order to provide additional management and monitoring tools. With an intuitive user interfaces 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 B—The Syscalls Intercepted from the Attacking Program

The syscalls in this appendix were obtained by employing 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 accessed 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 firework"], [/* 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 C—References

[1] R. Uhlig, G. Neiger, D. Rodgers, A. L. Santoni, F. C. Martins, A. V. Anderson, S. M. Bennett, A. Kagi, F. H. Leung and L. Smith, Intel virtualization technology, Computer, 38(5), pp.48-56, 2005.

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

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

[4] D. Perez-Botero, J. Szefer and R. B. Lee, Characterizing hypervisor vulnerabilities in cloud computing servers, In Proceedings of the 2013 International Workshop on Security in Cloud Computing (pp. 3-10). ACM, May 2013.

[5] A. Thongthua and S. Ngamsuriyaroj, Assessment of hypervisor vulnerabilities, In International Conference Cloud Computing Research and Innovations (ICCCRI), pp. 71-77, May 2016.

[6] J. Szefer, E. Keller, R. B. Lee and J. Rexford, Eliminating the hypervisor attack surface for a more secure cloud, In Proceedings of the 18th ACM Conference on Computer and Communications Security (pp. 401-412). ACM, Oct 2011.

[7] M. Graziano, A. Lanzi and D. Balzarotti, Hypervisor memory forensics, In International Workshop on Recent Advances in Intrusion Detection (pp. 21-40). Springer, Berlin, Heidelberg. 2013.

[8] L. M. Joshi, M. Kumar and R. Bharti, Understanding threats in hypervisor, its forensics mechanism and its research challenges, International Journal of Computer Applications 119.1 (2015).

[9] G. J. Popek and R. P. Goldberg, Formal requirements for virtualizable third generation architectures, Communications of the ACM 17.7 (1974): 412-421.

[10] B. Pariseau, KVM reignites Type 1 vs Type 2 hypervisor debate, retrieved from https://searchservervirtualization.techtarget.com/news/2240034817/KVM-reignites-Type-1-vs-Type-2-hypervisor-debate on Apr-11-2018.

[11] Xen project software overview, retrieved from https://wiki.xen.org/wiki/Xen_Project_Software_Overview on Apr-11- 2018.

[12] KVM, retrieved from https://www.linux-kvm.org/page/Main_Page on Apr-12-2018.

[13] J. Shi, Y. Yang and C. Tang, Hardware assisted hypervisor introspection, SpringerPlus, vol. 5, no. 1, 2016.

[14] NIST National Vulnerability Database, retrieved from https://nvd.nist.gov on Apr-12-2018.

[15] QEMU--the FAST! processor emulator, retrieved from https://www.qemu.org on Apr-12-2008.

[16] J.F. Kloster, J. Kristensen and A. Mejlholm, Efficient memory sharing in the Xen virtual machine monitor, Department of Computer Science, Aalborg University (Jan. 2006).

[17] X86 paravirtualised memory management, retrieved from https://wiki.xen.org/wiki/X86_Paravirtualised_Memory_Management.

[18] Y. Song, H. Wang and T. Soyata, Hardware and software aspects of VM-based mobile-cloud offloading, Enabling Real-Time Mobile Cloud Computing through Emerging Technologies, pp.247-271, 2015.

[19] A. Belay, A. Bittau, A. J. Mashtizadeh, D. Terei, D. Mazieres and C. Kozyrakis, Dune: safe user-level access to privileged CPU features, October 2012, In Osdi (Vol. 12, pp. 335-348).

[20] Pandavirtualization: exploiting the Xen hypervisor, Retrieved from https://googleprojectzero.blogspot.com/2017/04/pandavirtualization-exploiting-xen.html on May-30-2018.

[21] Xen exploitation part 3: XSA-182, Qubes escape, Retrieved from https://blog.quarkslab.com/xen-exploitation-part-3-xsa-182-qubes-escape.html.

[22] J. Satran, L. Shalev, M. Ben-Yehuda and Z. Machulsky, Scalable I/O-a well-architected way to do scalable, secure and virtualized I/O, In Workshop on I/O Virtualization, Dec 2008.

[23] 2015 state of hyperconverged infrastructure market report, Retrieved from https://www.actualtechmedia.com/wp-content/uploads/2015/05/2015-State-of-Hyperconverged-Infrastructure-Market-Report.pdf on Aug-3-2018.

[24] C. Liu, A. Singhal and D. Wijesekera, A layered graphical model for mission attack impact analysis, In Communications and Network Security (CNS), Oct, 2017 IEEE Conference on (pp. 602-609).

[25] H. Fayyad-Kazan, L. Perneel and M. Timmerman, Full and para-virtualization with Xen: a performance comparison. Journal of Emerging Trends in Computing and Information Sciences, 4(9), 2013.