Determining Forensic Data Requirements for Detecting Hypervisor Attacks

Changwei Liu[1], Anoop Singhal[2], Ramaswamy Chandramouli[2], and Duminda Wijesekera[1,2]

[1]Department of Computer Science, George Mason University, Fairfax VA 22030 USA

[2]National Institute of Standards and Technology, 100 Bureau Drive, Gaithersburg MD 20899 USA

[1]{*cliu6,dwijesek*}*@gmu.edu*

[2]{*anoop.singhal,ramaswamy.chandramouli*}*@nist.gov*

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. However, hypervisors are complex software systems with many lines of code and known to have vulnerabilities. This paper 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, with the objective of determining the evidence coverage for detecting those attacks and identifying techniques required to gather missing evidence to reconstruct the attacks.

Keywords: cloud computing, forensic analysis, hypervisors, Xen, KVM, vulnerabilities

1 Introduction

Most cloud services are provided using virtualized environments. 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]. Although Operating Systems (OS) directly control hardware resources, virtualization, typically provided using a hypervisor (a.k.a. a virtual machine monitor (VMM)) [3] within a cloud environment, provides an abstraction of hardware and system resources. As a software layer that lies between the physical hardware and the Virtual Machines (VMs or guest machines), a hypervisor supports the guest virtual machines by presenting the guest OSs with a virtual operating platform and managing their execution. However, hypervisors are complex software systems with many lines of code and known vulnerabilities [4]. By exploiting these vulnerabilities, attackers can possess the accessibility and authorization over the hypervisors, and subsequently attack all VMs running under the compromised hypervisor.

Many researchers have characterized hypervisor vulnerabilities, assessed them, created tools to detect them, or identified evidence for attack forensic analysis [4–8, 30]. Hypervisor forensics aims at extracting left over post-attack artifacts in order to investigate and analyze attacks at the hypervisor level. Though work such as inspecting the physical memory to find evidence has been broadly explored [7], there isn't much work that analyzes the recent hypervisor vulnerabilities to derive a profile of hypervisor vulnerabilities and uses them to discover forensic evidence that can reconstruct hypervisor attacks. Motivated by the work presented in [4] that characterized hypervisor vulnerabilities as of July 2012 with the objective of preventing their exploitation, in this work, we analyzed the recent vulnerability reports associated with two popular open-source hypervisors, Xen and KVM in the National Institute of Standards and Technology's National Vulnerability Database (NIST-NVD), with the objective of discovering recent trends in hypervisor attacks, deriving a profile of recent hypervisor attacks, determining the forensic evidence coverage and identifying techniques to collect supporting evidence in order to reconstruct hypervisor attacks. The main

contributions of this paper are as follows: (1) Analyzing and classifying recently reported hypervisor vulnerabilities of Xen and KVM (between 2016 and 2017) in the NIST-NVD database, and deriving a profile of recent hypervisor vulnerabilities based on hypervisor functionalities, attack types and sources of attacks; (2) Simulating some sample attacks representing the attack trends, ascertaining their forensic data coverage and exploring methods that can identify the evidence to reconstruct hypervisor attacks.

The rest of this paper 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 of Xen and KVM 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 OSs. 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 paper 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].

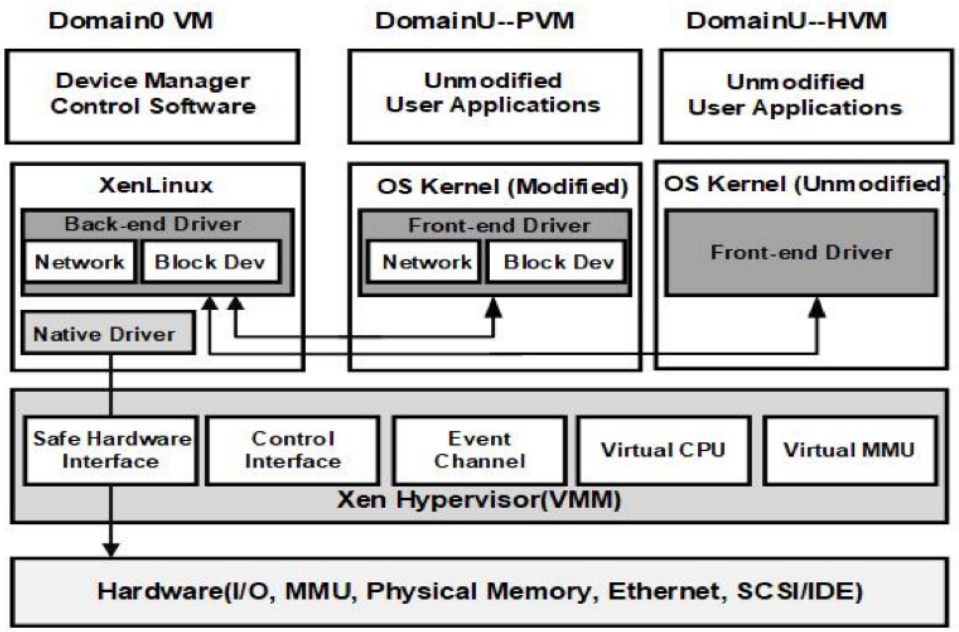

Figure 1: The Xen architecture

2.1.2 KVM

KVM was first introduced in 2006 and soon merged into the Linux kernel (2.6.20) among open-source hypervisor projects. 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.

2.2 Related Work

Hypervisor attacks are categorized as external attacks and defined as exploits of the hypervisor's vulnerabilities that enable attackers to gain accessibility to and authorization over 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,

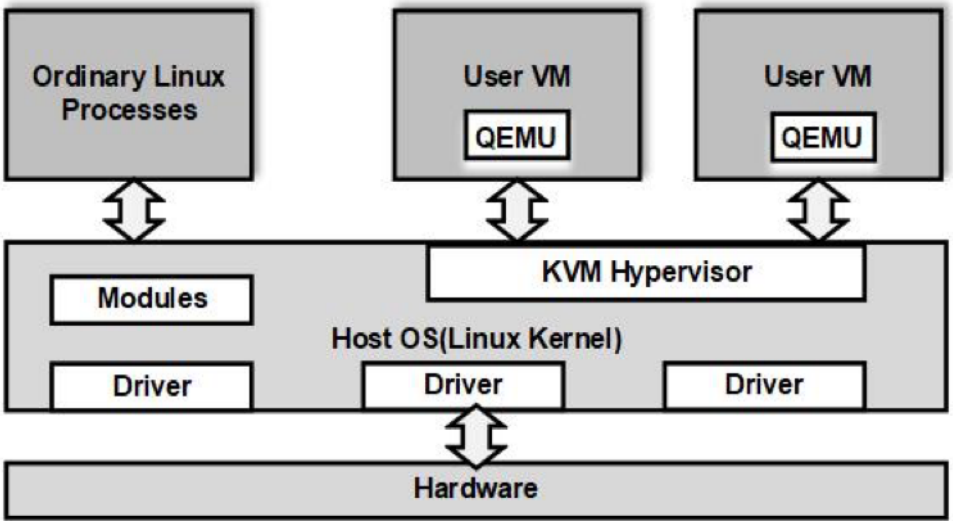

Figure 2: The KVM architecture

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], proposed or developed tools using VM introspection from the hypervisor level to detect attacks in VMs [26, 30], and leveraged existing memory forensic techniques to explore the technique that detects the existence and the characteristics of any hypervisor in a host memory dump [7].

3 Deriving a Profile of Hypervisor Vulnerabilities

We use the following criteria to derive a profile of recent hypervisor vulnerabilities as a prelude for determining forensic data requirements for detecting hypervisor attacks:

- 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 used 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 provided.

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 three classification criteria: hypervisor functionality, attack type and attack source.

3.2 Hypervisor Functionality

To better understand different hypervisor vulnerabilities, Perez-Botero et al. considered 11 functionalities that a traditional hypervisor provides and mapped vulnerabilities to them [4]: (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 4 and 5 above, these were merged into a single functionality. A detailed description of all these functionalities is as follows.

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 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) that enables multiple vCPUs belonging to the same VM to be scheduled on 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 a virtualized environment, the hypervisor emulates the MMU (therefore called the soft MMU) for the guest OS 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 a hypervisor provides I/O services to guest VMs. In Xen, they are: (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; (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; (3) the host assigns a device (known as a pass-through device) directly to the guest VM. In addition, in order to reduce I/O virtualization overhead and improve virtual machine performance, scalable self-virtualizing I/O devices that allow direct access interface to multiple VMs are also used.

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 buffers in a guest domain's memory and share the I/O buffer with Dom0 properly, which allows the split device drivers with 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 that is architecturally similar to Xen paravirtualized device drivers 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' 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 used similar to syscalls 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 VM non-root mode to VM root mode is called VMExit. VMExit is a response to some instructions

and events (e.g., page fault) from guest VMs and is 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 functionality including starting, pausing, or stopping VMs and are implemented in Xen Dom0 and the KVM's libvirt driver.

Remote Management Software: Remote management software is used as an interface that connects directly to the hypervisor in order to provide additional management and monitoring tools. With an intuitive user interface that visualizes the status of a system, the remote management software allows administrators to change 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.

3.3 Deriving a Vulnerability Profile

Based on the description listed in NIST-NVD, the 83 Xen and 20 KVM vulnerabilities during the years 2016 and 2017 were mapped to the ten hypervisor functionalities. Furthermore, with the goal of deriving a hypervisor security vulnerability profile, they were analyzed and classified according to functionalities, attack types (impacts), and attack sources.

Classifications based on the hypervisor functionalities are shown in Table 1. With the exception of Virtual Symmetric Multiprocessing and Remote Management Software, all other 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 the broader user base for Xen. Furthermore, approximately 69% of the vulnerabilities in Xen and 45% of the vulnerabilities in KVM are concentrated

Table 1: The vulnerabilities of Xen and KVM classified by functionalities

Number	Hypervisor Vulneraiblity	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%) 5 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%)
8	VM Management	8(10%)	0(0%)
9	Remote Management Software	0(0%)	0(0%)
10	Hypervisor Add-ons	0(0%)	1(5%)

Table 2: 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%)

in two functionalities—Soft MMU, 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, we associated the vulnerabilities based on the I/O and Networking functionality with each of the four types of I/O virtualization: fully virtualized devices, paravirtualized devices, direct access devices, and self-virtualized devices, which shows that most of the I/O and Networking vulnerabilities in Xen came from paravirtualized devices, and all I/O and networking vulnerabilities in KVM came from fully-virtualized devices. Our analysis shows that this is caused by the fact that in most Xen deployments, I/O and Networking functionality is configured using a paravirtualized device, while this functionality in KVM is configured using a fully virtualized device.

Table 3: The 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 (authenticated remote guest user) (5%)
Host OS user	0 (0%)	1 (5%)

Classifications based on the attack types and the sources of attacks are listed in Table 2 and Table 3. Table 2 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 the 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 3 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 monitor guest users' activities in order to reduce attack risks.

4 Sample Attacks and Forensic Data

Because the hypervisor vulnerability profile shows numerous vulnerabilities are related to Xen soft MMU functionality with the attack sources from the guest VMs, we launched two sample attacks, including those that exploit vulnerabilities CVE-2017-7228 and CVE-2016-6258, to explore the evidence coverage and the methodologies used to identify evidence in order to reconstruct the hypervisor attacks.

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) that support two different virtualization modes: Paravirtualization (PV) and Hardware-assisted Virtualization (HVM). The PV module has been widely utilized for its higher performance, according to our analysis in Table 1 and in [25]. However, because the Xen PV model uses complex code to emulate the MMU, it introduces many vulnerabilities, including CVE-2017-7228 and CVE-2016-6258.

CVE-2017-7228 was first reported by 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.

CVE-2016-6258 was reported by 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. By exploiting a vulnerability in the 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.

We launched both attacks on the PV module configured in Qubes 3.1 and Debian 8 with Xen 4.6. Figure 3 illustrates the attack experiment of using CVE-2017-7228 in Qubes 3.1, where the attacker impersonating the PV guest root user (the bottom green terminal is the attacker's VM that is named as *attacker*) executed command *qvm-run victim firefox* that could only be executed by Dom0 to run the victim PV

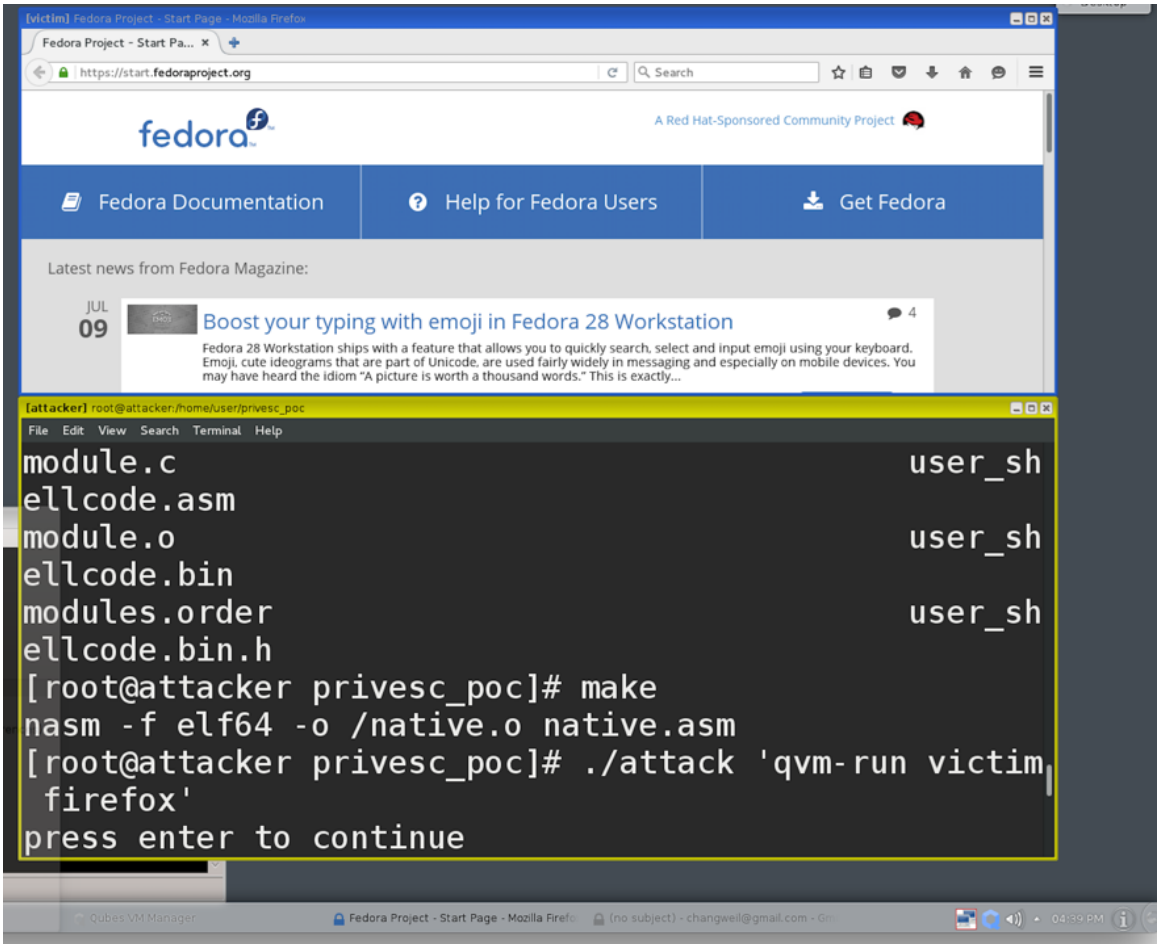

Figure 3: The CVE-2017-7228 and CVE-2016-6258 attacks

guest's *Firefox* web browser, which opened a webpage in the victim's VM(the webpage is shown as the upper blue window. In Qubes 3.1, *qvm-run* is the command that can only be executed by Dom0 to run an applications in a guess VM. *victim* is the name of the victim guest VM in our experiment). As illustrated in Figure 3, both attacks allowed the PV guest users to gain the control of Dom0.

4.2 Identifying Evidence Coverage for Forensic Analysis

Both attacks were launched from the guest VMs, which 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 logged to perform a forensic analysis. An Analysis of the device activity logs and runtime syscalls showed the relevant evidence originated from the syscalls obtained from the attackers' VMs. Figure 4 illustrates

syscalls obtained from the attacking process in the attacker's VM (we omit some of them due to space limitations), showing that (1) the attacker executed the attack program with arguments *qvm-run victim firefox* aimed at the victim guest's VM (Line 1); (2) the attack program and required Linux libraries have been loaded to the memory for the program execution (Line 2 to 4); (3) the memory pages of the attack program have been protected from being accessed by other processes (Line 5 to Line 8); and (4) the attack program injected a loadable Linux module named *test.ko* to the kernel space to exploit the vulnerability and deleted it afterwards (Line 9 to Line 15). Despite the noise among syscalls that can be found in most programs, other syscalls, such as Line 1, Line 9 to Line 15, 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.

The acquisition of relevant and admissible evidence plays an important role in forensic analysis by determining and reconstructing attacks. As presented in [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 from the attacker's VM 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 can be easily compromised by the attacker. Thus, approaches that enable monitoring VMs from the hypervisor can get more supporting and admissible evidence.

4.3 Using Virtual Machine Introspection (VMI) for Hypervisor Forensics

The VMI can be used to inspect the states of a VM from either a privileged VM or the hypervisor, for the purpose of analyzing the software running inside the VM [26].

```
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.      …
5.      mprotect(0x7fa3a2df9000, 16384, PROT_READ) = 0
6.      mprotect(0x600000, 4096, PROT_READ) = 0
7.      mprotect(0x7fa3a3023000, 4096, PROT_READ) = 0
8.      …
9.      open("test.ko", O_RDONLY) = 3
10.     finit_module(3, "user_shellcmd_addr=1407334317317"..., 0) = 0
11.     fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 0), ...}) = 0
12.     mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
 = 0x7fa3a3021000
13.     mmap(0x600000000000, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|M
AP_ANONYMOUS|MAP_LOCKED, -1, 0) = 0x600000000000
14.     delete_module("test", O_NONBLOCK) = 0
15.     exit_group(0)= ?
```

A brief description about these syscalls is provided below to help with understanding:
execve() executes the program pointed by the first argument;
brk() changes the location of the program break, which defines the end of the process' data segment;
mmap() creates a new mapping in the virtual address space of the calling process;
mprotect() changes the access protections for the calling process' memory pages;
open() opens a file, **test.ko** in our case;
finit_module() loads a kernel module, **test.ko** in our case;
fstat() gets the file status, and the first argument is the file descriptor;
delete_module() unloads the injected module;
exit_group() exits all threads in a process.

Figure 4: The syscalls intercepted from the attacking program

The states 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. VMI-based applications using this capability can be built for forensic analysis in the following ways:

- 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 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].

- A VMI-based application can be built to analyze execution paths of the monitored VM by tracing the sequence of VM activities and the corresponding com-

plete VM state (e.g., memory map, IO access). This can be used to construct a detailed attack graph with the VM states as nodes and the VM activities as edges, thereby tracing the path through which the current compromised state was reached [31].

Though VMI addresses deficiencies in forensic analysis that simply uses the system calls from the compromised VM, VMI tools must reconstruct operational semantics of the guest operating system based on low-level sources such as physical memory and CPU registers [27]. Because LibVMI [28] 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) [28] installed on Dom0, we captured all running processes and injected Linux modules of the guest attack VM. We illustrated the ones captured during the attack time of CVE-2017-7228 in Figure 5, in which Line 1 and Line 10 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; Line 3 to 9 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 Line 11 to 16 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, it was easy to identify the attack process (named as *attack* in Line 9) and the injected attack module (named as *test* in Line 11. The 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, in order to access the consistent memory access, it pauses and resumes the guest VM. e.g., our experiment showed that the attacker's VM was paused for about 0.035756 second and 0.036173 second for

```
1. root@debian:/home/guest/src/libvmi/libvmi# ./examples/vmi-process-list pv-attacker
2. Process listing for VM pv-attacker (id=2)
3. [    0] swapper/0 (struct addr:ffffffff81e13500)
4. [    1] systemd (struct addr:ffff88007c460000)
5. …
6. [  674] (sd-pam) (struct addr:ffff880076104600)
7. [  677] bash (struct addr:ffff880003c8aa00)
8. [  703] sudo (struct addr:ffff880004341c00)
9. [  704] attack (struct addr:ffff880004343800)

10.root@debian:/home/guest/src/libvmi/libvmi# ./examples/vmi-module-list pv-attacker
11.test
12.intel_rapl
13.x86_pkg_temp_thermal
14.Coretemp
15.crct10dif_pclmul
16.…
```

Figure 5: The running processes and injected modules from the attacker's VM

capturing the running process and injected modules. 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 [29]. Third, storing the captured snapshots of the guest VMs for forensic analysis often requires a large portion 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 Conclusion

In this paper, we analyzed and classified all reported vulnerabilities on Xen and KVM in the last two years, which derived a profile of the recent hypervisor vulnerabilities in the two chosen hypervisors, showing 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 and most attackers are guest OS users. Based on the hypervisor vulnerability profile, we launched two sample attacks for forensic analysis. The collected data on the sample attacks showed that most valuable evidence remains in the run-time system memory. Therefore, in order to obtain valuable evidence with guaranteed integrity, VM introspection technique that inspects the state of the guest VMs from the hyper-

visor level that provides strong isolation from the guest VM should be implemented and used.

Disclaimer

This paper is not subject to copyright in the United States. Commercial products are identified in order to adequately specify certain procedures. In no case does such an identification imply a recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the identified products are necessarily the best available for the purpose.

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), 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), 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 on Sep-25-2018.

[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, In Osdi (Vol. 12, pp. 335-348), Oct 2012.

[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 on May-30-2018.

[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 cloud forensic mission attack impact Analysis, 14th IFIP WG 11.9 International Conference, New Delhi, India, January 3-5, 2018.

[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.

[26] T. Garfinkel and M. Rosenblum, A Virtual Machine Introspection Based Architecture for Intrusion Detection, In Ndss (Vol. 3, No. 2003, pp. 191-206), Feb 2003.

[27] B. Dolan-Gavitt, B. Payne and W. Lee, Leveraging forensic tools for virtual machine introspection, Georgia Institute of Technology, 2011.

[28] LibVMI—Virtual Machine Introspection, retrieved from http://libvmi.com, Sep 2018.

[29] G. Wang, J. E. Zachary, C. M. Pham, Z. T. Kalbarczyk, and R. K. Iyer, Hypervisor Introspection: A Technique for Evading Passive Virtual Machine Monitoring, In WOOT, 2015.

[30] B. D. Payne, Simplifying virtual machine introspection using libvmi. Sandia report, pp.43-44, 2012.

[31] A. Moser, C. Kruegel and E. Kirda, Exploring multiple execution paths for malware analysis. In Security and Privacy, SP'07. IEEE Symposium on (pp. 231-245), May 2007.