Determining the Forensic Data Requirements for Investigating Hypervisor Attacks

Changwei Liu, Anoop Singhal, Ramaswamy Chandramouli, Duminda Wijesekera

Chapter 14

DETERMINING THE FORENSIC DATA REQUIREMENTS FOR INVESTIGATING HYPERVISOR ATTACKS

Changwei Liu, Anoop Singhal, Ramaswamy Chandramouli and Duminda Wijesekera

Abstract Hardware/server virtualization is commonly employed in cloud computing to enable ubiquitous access to shared system resources and provide sophisticated services. The virtualization is typically performed by a hypervisor, which provides mechanisms that abstract hardware and system resources from the operating system. However, hypervisors are complex software systems with many vulnerabilities. This chapter analyzes recently-discovered vulnerabilities associated with the Xen and KVM open-source hypervisors, and develops their attack profiles in terms of hypervisor functionality (attack vectors), attack types and attack sources. Based on the large number of vulnerabilities related to hypervisor functionality, two sample attacks leveraging key attack vectors are investigated. The investigation clarifies the evidence coverage for detecting attacks and the missing evidence needed to reconstruct attacks.

Keywords: Cloud computing, hypervisors, Xen, KVM, vulnerabilities, forensics

1. Introduction

Most cloud services are provided by 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 [15, 28]. Although an operating system directly controls hardware resources, virtualization via a hypervisor or virtual machine monitor (VMM) [6] in a cloud environment provides an abstraction of hardware and system resources. The hypervisor serves as a software layer between the physical hardware and virtual (or

guest) machines, presenting the guest operating systems with virtual operating platforms and managing their execution. However, hypervisors are complex software systems with numerous lines of code and many vulnerabilities [20]. These vulnerabilities can be exploited to gain access to and control a hypervisor, and subsequently attack all the virtual machines that execute in the compromised hypervisor.

Several researchers have characterized and assessed hypervisor vulnerabilities, developed tools for detecting vulnerabilities and identified evidence that can be used for attack forensics [7, 9, 19, 20, 26, 27]. Hypervisor forensics seeks to extract leftover artifacts in order to investigate and analyze attacks at the hypervisor level. Techniques such as inspecting physical memory to locate evidence of attacks have been explored [7], but little, if any, research has analyzed recent hypervisor vulnerabilities to derive attack profiles and leverage them to discover forensic evidence that can help reconstruct hypervisor attacks.

The research described in this chapter was motivated by the work of Perez-Botero et al. [20] that characterized hypervisor vulnerabilities with the objective of preventing their exploitation. This chapter focuses on recent (2016 and 2017) vulnerability reports associated with the popular, open-source Xen and KVM hypervisors, which are listed in the National Institute of Standards and Technology National Vulnerability Database (NIST-NVD). The chapter analyzes and classifies the vulnerabilities to derive attack profiles based on hypervisor functionality, attack type and source. Additionally, it simulates sample attacks to ascertain their forensic data coverage and explore methods for identifying and obtaining the evidence needed to reconstruct attacks.

2. Background and Related Work

This section discusses hypervisor architectures with a focus on the Xen and KVM hypervisors. Also, it discusses related work in the area of cloud forensics.

2.1 Hypervisors

Hypervisors are software and/or firmware modules that virtualize system resources such as CPU, memory and devices. Popek and Goldberg [21] have classified hypervisors as Type 1 and Type 2 hypervisors. A Type 1 hypervisor runs directly on the host hardware to control the hardware and manage guest operating systems. For this reason, a Type 1 hypervisor is sometimes called a "bare metal" hypervisor. Example Type 1 hypervisors are Xen, Microsoft Hyper-V and VMware ESX/ESXi.

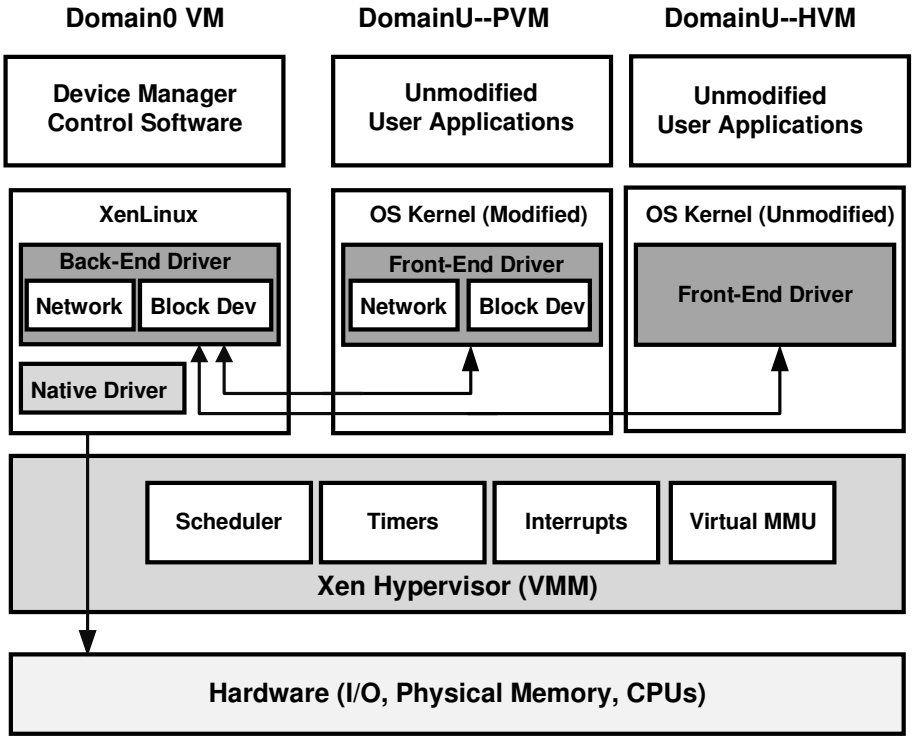

Figure 1. Xen hypervisor architecture.

A Type 2 hypervisor is similar to a program that executes as a process in an operating system. Example Type 2 hypervisors are VMware Player, VirtualBox, Parallels Desktop for Mac and QEMU.

Some hypervisors have features of Type 1 and Type 2 hypervisors. For example, the Linux-kernel-based virtual machine (KVM) is a kernel module that effectively converts the host operating system to a Type 1 hypervisor, but it is also categorized as a Type 2 hypervisor because a Linux distribution is a general-purpose operating system that executes other applications that compete for virtual machine resources [18].

A market report [14] lists the most popular hypervisors as Microsoft Hyper-V, VMware VSphere/ESX, Citrix XenServer/Xen and KVM. Because Microsoft Hyper-V and VMware VSphere/ESX are commercial products, this work focuses on the remaining two hypervisors, Xen and KVM, which are both open source.

Xen Hypervisor. Figure 1 shows the Xen hypervisor architecture. The hypervisor manages three kinds of virtual machines. The first is the control domain (Dom0) and the other two are guest domains (DomU)

that support two virtualization modes – paravirtualization (PV) and hardware-assisted virtualization (HVM) [31]. Dom0 is the initial domain started by the Xen hypervisor upon booting up a privileged domain. It plays the administrator role and provides services to the DomU virtual machines.

Paravirtualization in a DomU guest domain is a highly efficient and lightweight virtualization technology introduced by Xen that does not require virtualization extensions from the host hardware. Thus, paravirtualization enables virtualization on a hardware architecture that does not support hardware-assisted virtualization; however, it requires paravirtualization-enabled kernels and paravirtualization drivers in order to power a high performance virtual server.

Hardware-assisted virtualization requires hardware extensions. Xen typically uses QEMU (Quick Emulator) [22], a generic hardware emulator that simulates personal computer hardware (e.g., CPU, BIOS, IDE, VGA, network cards and USBs). Because of the use of simulation technologies, the performance of a virtual machine with hardware-assisted virtualization is inferior to that of a paravirtualization virtual machine.

Xen 4.4 provides the new PVH virtualization mode with lightweight hardware-assisted-virtualization-like guests that use virtualization extensions in the host hardware. Unlike hardware-assisted virtualization guests that (for example) use QEMU to emulate devices, PVH guests use paravirtualization drivers for input/output and native operating system interfaces for virtualized timers, virtualized interrupts and booting. A PVH guest also requires a PVH-enabled guest operating system [31].

KVM Hypervisor. KVM was introduced in 2006 and it was soon merged into the Linux kernel (2.6.20) in open-source hypervisor projects. KVM is a full virtualization solution for Linux that runs on x86 hardware with virtualization extensions (Intel VT or AMD-V) to enable virtual machines to execute as normal Linux processes [11].

Figure 2 shows a KVM hypervisor architecture that uses QEMU to create guest virtual machines that execute as separate user processes. KVM is considered to be a Type 2 hypervisor because it is installed on top of the host operating system. However, the KVM kernel module turns the Linux kernel into a Type 1 bare-metal hypervisor, providing the functionality of the most complex and powerful Type 1 hypervisors.

2.2 Related Work

Hypervisor attacks are defined as exploits of hypervisor vulnerabilities that enable external attackers to gain access to and control hyper-

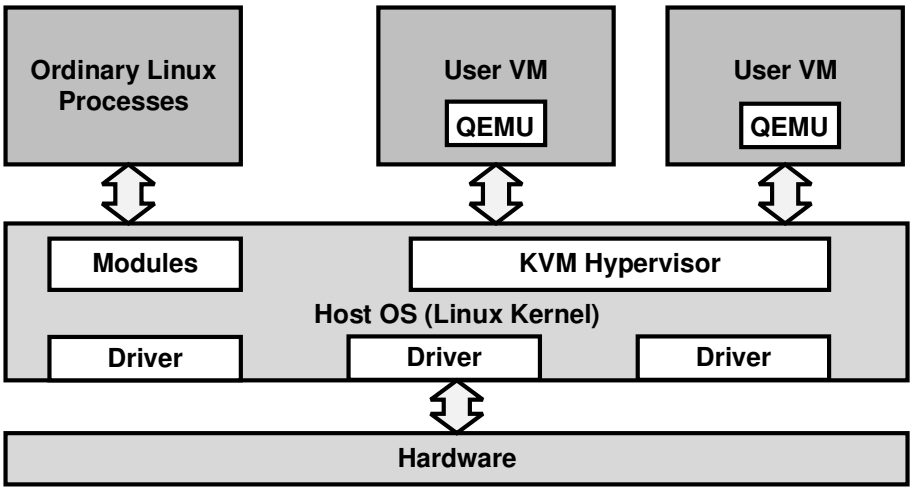

Figure 2. KVM hypervisor architecture.

visors [24]. Perez-Botero et al. [20] have characterized Xen and KVM vulnerabilities based on hypervisor functionality. However, the characterizations cannot be used to predict attack trends.

Thongthua and Ngamsuriyaroj [27] have assessed the vulnerabilities of popular hypervisors, including VMware ESXi, Citrix XenServer and KVM, using the NIST 800-115 security testing framework. Their assessments of vulnerabilities cover weaknesses, severity scores and attack impacts.

Joshi et al. [9] have researched threats to hypervisors and hypervisor forensic mechanisms. They also discuss the use of virtual machine introspection at the hypervisor level to detect attacks [5, 19] and memory forensic techniques to identify hypervisor attack artifacts in a host memory dump [7].

3. Deriving Hypervisor Attack Profiles

As a prelude to determining the forensic data requirements for detecting hypervisor attacks, the following criteria are employed to derive attack profiles based on recent hypervisor vulnerabilities:

- **Hypervisor Functionality:** The specific functionality that leads to the existence of a vulnerability (attack vector).

- **Attack Type:** The specific impact of exploiting a vulnerability.

- **Attack Source:** The specific component in a hypervisor platform where an attack is launched.

The Xen and KVM hypervisor attack profiles were derived by first identifying all the vulnerabilities in the NIST-NVD that were posted during 2016 and 2017. Each 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 NIST-NVD vulnerability descriptions. The total number of vulnerabilities in each of the three categories (attack vector, attack type and attack source) thus constitute the recent attack profiles for the two hypervisors.

3.1 NIST-NVD Vulnerabilities

The NIST-NVD is a repository of standards-based vulnerability management data, which includes databases of security checklist references, security-related software flaws, misconfigurations, product names and impact metrics [17]. A search of the NIST-NVD for vulnerabilities posted in 2016 and 2017 revealed 83 Xen hypervisor vulnerabilities and 20 KVM hypervisor vulnerabilities. These vulnerabilities were associated with the three classification criteria: (i) hypervisor functionality; (ii) attack type; and (iii) attack source.

3.2 Hypervisor Functionality

In order to better understand hypervisor vulnerabilities, Perez-Botero et al. [20] considered eleven traditional hypervisor functionalities and mapped vulnerabilities to them. The eleven vulnerabilities are: (i) virtual CPUs; (ii) virtual symmetric multiprocessing; (iii) soft memory management units; (iv) input/output and networking; (v) paravirtualized input/output; (vi) interrupt and timer mechanisms; (vii) hypercalls; (viii) VMExits; (ix) virtual machine management; (x) remote management software; and (xi) hypervisor add-ons. Based on the common functions provided by functionalities (iv) and (v), the two were merged as a single functionality.

- **Virtual CPUs:** A virtual CPU (also called a virtual processor) abstracts a share of a physical CPU assigned to a virtual machine. The hypervisor allocates a portion of the physical CPU cycles to a virtual CPU assigned to a virtual machine. The hypervisor also schedules virtual CPU tasks to the physical CPUs.

- **Virtual Symmetric Multiprocessing:** Virtual symmetric multiprocessing enables multiple virtual CPUs belonging to the same

virtual machine to be scheduled on a physical CPU with at least two logical processors.

- **Soft Memory Management Units:** A memory management unit (MMU) is the hardware that manages memory by translating the virtual addresses manipulated by software to physical addresses. In a virtualized environment, the hypervisor emulates the memory management unit – called the soft memory management unit – of a guest operating system by mapping what the guest operating system sees as physical memory (called pseudo-physical/physical addresses in Xen) to the underlying memory of the machine (called machine addresses in Xen). The physical address to machine address mapping table is typically maintained in the hypervisor and hidden from a guest operating system using a shadow page table for the guest virtual machine. Each shadow page table mapping, which translates virtual addresses of programs in a guest virtual machine to guest (pseudo) physical addresses, is placed in the guest operating system [10, 30].

  The Xen paravirtualized memory management unit model requires a guest operating system to be directly aware of the mapping between (pseudo) physical and machine addresses (using a P2M table). Additionally, in order to read page table entries that contain machine addresses and convert them back to (pseudo) physical addresses, translation from machine to (pseudo) physical addresses via a reverse M2P table is required by the Xen paravirtualized memory management unit model [30].

- **Input/Output and Networking:** A hypervisor provides input/output services to guest virtual machines via three common approaches (all of which are employed by Xen): (i) the hypervisor emulates a known input/output device in a fully virtualized system and each guest uses a native unmodified driver to interact with it; (ii) a paravirtual front-end driver in a paravirtualized system is installed in a modified guest operating system in DomU, which uses shared-memory asynchronous buffer-descriptor rings to communicate with the back-end input/output driver in the hypervisor; and (iii) the host assigns a pass-through device directly to the guest virtual machine. Scalable self-virtualizing input/output devices that allow direct access interface to multiple virtual machines are also employed to reduce input/output virtualization overhead and improve virtual machine performance.

  Although hypervisors enforce isolation across virtual machines residing in a single physical machine, Xen uses the grant mecha-

nism for inter-domain communications. Shared-memory communications between unprivileged domains are implemented via grant tables [10]. Grant tables protect the input/output buffers in guest domain memory and share the buffers with Dom0, which enable split device drivers with block device and network interface card input/output. Each domain has its own grant table that allows the domain to inform Xen about the permissions that other domains have on their pages.

KVM typically uses Virtio, a virtualization standard for network and disk drivers. Virtio is architecturally similar to Xen paravirtualized device drivers that comprise front-end and back-end drivers.

- **Interrupt and Timer Mechanisms:** Hypervisors should be able to virtualize and manage interrupts and timers [25], interrupt/timer controllers of guest operating systems and guest operating system accesses to controllers. The interrupt and timer mechanisms in a hypervisor include a programmable interval timer, advanced programmable interrupt controller and interrupt request mechanisms [20].

- **Hypercalls:** Hypercalls are similar to system calls (syscalls) that provide user-space applications with kernel-level operations. They are used like syscalls with up to six arguments passed in registers. A hypercall layer, which is commonly available, enables guest operating systems to make requests to the host operating system. Domains use hypercalls to request privileged operations from hypervisors such as updating page tables. As a result, an attacker can use hypercalls to attack a hypervisor from a guest virtual machine.

- **VMExits:** Belay at el. [1] describe VMExits as changing virtual machines from the non-root mode to the root mode. VMExits are triggered by certain events in guest virtual machines – external interrupts, triple faults, task switches, input/output operation instructions (e.g., INB and OUTB) and control register accesses. VMExits are the main source of performance degradation in virtualized systems.

- **Virtual Machine Management:** Hypervisors support basic virtual machine management functionality, including starting, pausing and stopping virtual machines. They are implemented in Xen's Dom0 and KVM's `libvirt` driver.

- **Remote Management Software:** Remote management software serves as an interface that connects directly to a hypervisor,

Table 1. Xen and KVM vulnerabilities classified by functionality.

No.	Hypervisor Functionality	Xen	KVM
1	Virtual CPUs	6 (7%)	4 (20%)
2	Virtual Symmetric Multiprocessing	0 (0%)	0 (0%)
3	Soft Memory Management Units	34 (41%)	5 (25%)
4	Input/Output and Networking	24[1] (30%)	4[2] (20%)
5	Interrupt and Timer Mechanisms	7 (8%)	3 (15%)
6	Hypercalls	3 (4%)	1 (5%)
7	VMExits	1 (1%)	2 (10%)
8	Virtual Machine Management	7 (8%)	0 (0%)
9	Remote Management Software	1 (1%)	0 (0%)
10	Hypervisor Add-Ons	0 (0%)	1 (5%)

[1]Five are fully-virtualized; 19 are paravirtualized; none are direct access or self-virtualized
[2]All are fully-virtualized

providing additional management and monitoring tools. By providing an intuitive user interface that visualizes system status, remote management software enables an administrator to provision and manage virtualized environments.

- **Hypervisor Add-Ons:** Hypervisor add-ons use modular designs to add extended functions. By leveraging interactions between add-ons and a hypervisor, an attacker can cause a host to crash (denial-of-service attack) or even compromise the host.

3.3 Deriving Attack Profiles

Based on the descriptions posted in the NIST-NVD, the 83 Xen and 20 KVM vulnerabilities identified during 2016 and 2017 were mapped to the ten hypervisor functionalities. In order to derive the hypervisor attack profiles, the vulnerabilities were analyzed and classified according to functionality, attack type (impact) and attack source.

Table 1 shows the Xen and KVM vulnerabilities classified by functionality. The numbers of vulnerabilities and their percentages are listed for each hypervisor functionality. With the exception of virtual symmet-

Table 2. Types of attacks leveraging Xen and KVM vulnerabilities.

Attack Type	Xen	KVM
Denial-of-Service	48[1] (44%)	17[2] (63%)
Privilege Escalation	33[3] (30%)	3[4] (11%)
Information Leakage	15[5] (14%)	5 (19%)
Arbitrary Code Execution	8[6] (7%)	2[7] (7%)
Reading/Modifying/Deleting Files	3 (3%)	0 (0%)
Others (e.g., Host Compromise, Canceling Other Administrators' Operations and Data Corruption)	3 (3%)	0 (0%)

[1]Four have other impacts; [2]Three have other impacts; [3]Sixteen have other impacts
[4]Two have other impacts; [5]Five have other impacts; [6]Two have other impacts
[7]All have other impacts

ric multiprocessing, all the other functionalities were reported as having vulnerabilities.

The table reveals that Xen has more vulnerabilities than KVM; one reason may be Xen's broader user base. Furthermore, approximately 71% of the vulnerabilities in Xen and 45% of the vulnerabilities in KVM are concentrated in two functionalities – soft memory management units, and input/output and networking. A detailed analysis of CVE reports indicates that these vulnerabilities primarily originate in page tables and input/output grant table emulation.

Additionally, the vulnerabilities based on input/output and networking functionality were associated with each of the four types of input/output virtualization: fully virtualized devices, paravirtualized devices, direct access devices and self-virtualized devices. The associations revealed that most of the input/output and networking vulnerabilities in Xen come from paravirtualized devices and all the input/output and networking vulnerabilities in KVM come from fully-virtualized devices. In the case of Xen, this is because, in most Xen deployments, input/output and networking functionality is configured using a paravirtualized device. In the case of KVM, the functionality is configured using a fully virtualized device.

Table 2 shows the types of attacks that leverage vulnerabilities in Xen and KVM hypervisors. The most common attack type is denial-of-service (44% for Xen and 63% for KVM), indicating that attacking the availability of cloud services could be a serious cloud security problem.

Table 3. Numbers of attacks from various attack sources.

Attack Source	**Xen**	**KVM**
Cloud Administrators	2[1] (2%)	0 (0%)
Guest Operating System Administrators	17[2] (20%)	1 (5%)
Guest Operating System Users	63[3] (76%)	17[4] (85%)
Remote Users	1 (1%)	1[5] (5%)
Host Operating System Users	0 (0%)	1 (5%)

[1]Management; [2]Including HVM and PV administrators
[3]Including ARM, x86, HVM and PV users
[4]Including KVM L1, L2 and privileged users
[5]Authenticated remote guest users

The other top attacks are 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 these attacks occur with less frequency than denial-of-service attacks, they result in more serious damage by enabling attackers to obtain sensitive user information or compromise hosts or guest virtual machines.

Table 3 shows the numbers of attacks for various attack sources. The greatest source of attacks is guest operating system users (76% for Xen and 85% for KVM); other attack sources are cloud administrators, guest operating system administrators and remote users. This suggests that cloud providers should monitor guest user activities in order to reduce the risk of attacks.

4. Sample Attacks and Forensic Implications

Because the Xen hypervisor attack profile lists many vulnerabilities related to the soft memory management unit functionality and guest virtual machines as the major attack source, two sample attacks were executed to explore evidence coverage and methodologies for identifying evidence in order to reconstruct hypervisor attacks. One attack exploited the CVE-2017-7228 vulnerability and the other attack exploited the CVE-2016-6258 vulnerability.

4.1 Sample Attacks

As discussed earlier, the Xen hypervisor manages three kinds of virtual machines, the control domain (Dom0) and guest domains (DomU)

that support two virtualization modes, paravirtualization and hardware-assisted virtualization. The paravirtualization mode is popular due to its better performance [4]. However, because the Xen paravirtualization model uses complex code to emulate memory management units, vulnerabilities such as CVE-2017-7228 and CVE-2016-6258 are introduced.

The CVE-2017-7228 vulnerability was reported in 2017 [8]. This vulnerability in x86 64-bit Xen (versions 4.8.x, 4.7.x, 4.6.x, 4.5.x and 4.4.x) was caused by inadequate checking of the `XENMEM_exchange` function, which enables a paravirtualization guest user to access hypervisor memory outside the memory provisioned to the guest virtual machine. Therefore, a malicious 64-bit paravirtualization guest user who makes a hypercall `HYPERVISOR_memory_op` function invoke the `XENMEM_exchange` function may be able to access all the system memory, enabling a virtual machine escape from DomU to Dom0 (i.e., breaking out of the guest virtual machine and interacting with the hypervisor host operating system) to cause a hypervisor host crash or information leakage.

The CVE-2016-6258 vulnerability was reported in 2016 [2]. The paravirtualization module uses a page table to map pseudo-physical/physical addresses seen by a guest virtual machine to the underlying memory of the machine. Exploiting a vulnerability in the Xen paravirtualization page tables that enables unauthorized modifications to page table entries, a malicious paravirtualization guest can access the page directory with an updated write privilege and execute a virtual machine escape to break out of DomU and control Dom0.

The two attacks were executed on a paravirtualization module configured in Qubes 3.1 and Debian 8 with Xen 4.6. Figure 3 illustrates the attacks. The attacker impersonates a paravirtualization guest root user (the bottom terminal is the attacker's virtual machine (`attacker`)). The attacker executes the command `qvm-run victim firefox` to run the Firefox web browser in the paravirtualization guest user virtual machine (`victim`). This opens a webpage in the victim's virtual machine (shown in the upper window). Note that the `qvm-run` command in Qubes 3.1 can only be executed by Dom0 to run an application in a guest virtual machine. As shown in Figure 3, both the attacks enable a paravirtualization guest user to gain control of Dom0.

4.2 Identifying Evidence Coverage

The two attacks from guest virtual machines exploited hypercall and soft memory management unit vulnerabilities in Xen. In addition to using Xen's device activity logs, the runtime syscalls of the impacted

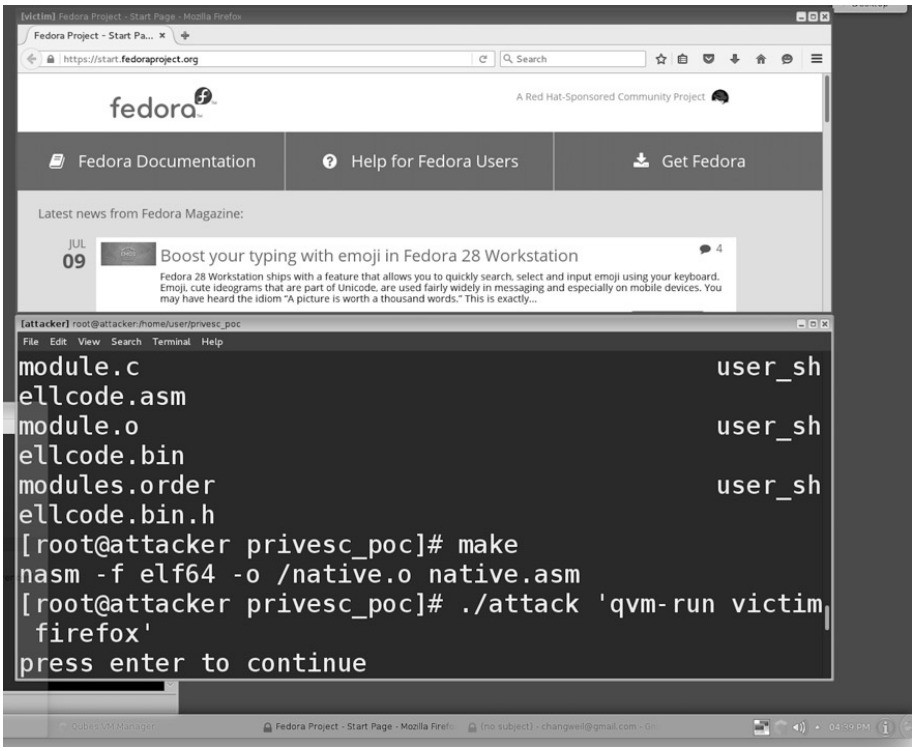

Figure 3. CVE-2017-7228 and CVE-2016-6258 attacks.

processes were logged. Subsequent analysis revealed that the syscalls obtained from the attacker's virtual machine constitute useful evidence.

Figure 4 shows the syscalls from the attacking process in the attacker's virtual machine (some of the syscalls are not presented due to space limitations). Specifically, the figure shows that:

- The attacker executed an attack program with the arguments `qvm-run victim firefox` targeting the victim's guest virtual machine (Line 1).
- The attack program and the required Linux libraries were loaded in memory for program execution (Lines 2–4).
- The memory pages of the attack program were protected from being accessed by other processes (Lines 5–8).
- The attack program injected the `test.ko` loadable Linux module into kernel space to exploit the vulnerability, and subsequently deleted the module (Lines 9–15).

```
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|MAP_ANONYMOUS|MAP_LOCKED, -1, 0) = 0x600000000000
14.  delete_module("test", O_NONBLOCK) = 0
15.  exit_group(0)= ?

     execve(): Executes the program pointed to 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 of the calling process
     memory pages
     open(): Opens the file test.ko
     finit\_module(): Loads the kernel module test.ko
     fstat(): Gets the file status; the first argument is the file
     descriptor
     delete\_module(): Unloads the injected module
     exit\_group(): Exits all the process threads
```

Figure 4. Syscalls intercepted from the attacking program.

Despite the noise in the syscalls (a common occurrence), other syscalls – such as those in Lines 1, 9 and 15 – reveal that the attack program injected a loadable kernel module into kernel space and proceeded to exploit the vulnerability to control Dom0. This opened the Firefox browser in the victim's guest virtual machine.

Clearly, the device activity logs and runtime syscalls constitute valuable evidence in a forensic investigation. Liu et al. [13] have shown that such evidence helps reconstruct attack paths in attack scenarios. Specifically, during the reconstruction process, an attack path with missing

attack steps drives the search for and collection of additional supporting evidence.

Analysis of the syscalls captured during the two sample attacks reveal that, while the syscalls obtained from the attacker's virtual machine are useful for forensic analysis, they lack attack details. In particular, the syscalls do not provide details of how features of the loadable kernel module used Xen's memory management to launch the attacks. Another deficiency is that the syscalls were collected from the attacker's guest virtual machine, which could have been easily compromised or manipulated by the attacker. Therefore, it is important to implement the monitoring of virtual machines from the hypervisor to obtain supporting evidence that would be admissible in legal proceedings.

4.3 Using Virtual Machine Introspection

Virtual machine introspection can be used to inspect the state of a virtual machine from a privileged virtual machine or the hypervisor to analyze the software running on the virtual machine [5]. The state information includes CPU state (e.g., registers), the entire memory and all input/output device states (e.g., contents of storage devices and the registers of input/output controllers).

The following virtual-machine-introspection-based forensic applications are promising:

- A virtual-machine-introspection-based application takes a snapshot of the entire memory and the input/output state of a victim's virtual machine. The captured state of the running victim virtual machine can be compared against a suspended virtual machine in a known good state or against the original virtual machine image from which the victim virtual machine was instantiated [5].

- A virtual-machine-introspection-based application analyzes execution paths of the compromised virtual machine by tracing the sequence of virtual machine activities and the corresponding complete virtual machine state (e.g., memory map, input/output access). A detailed attack graph is then constructed with virtual machine states as nodes and virtual machine activities as edges, helping trace the path leading to the compromised state [16].

Although virtual machine introspection addresses deficiencies in forensic analyses based on system calls from a compromised virtual machine, virtual machine introspection applications must reconstruct the operational semantics of the guest operating system based on low-level sources such as physical memory and CPU registers [3]. Because LibVMI [12]

```
1.  root@debian:/home/guest/src/libvmi/libvmi# ./examples/vmi-process-
    list pv-attacker
2.  Process listing for VM pv-attcker (id=2)
3.  [    0] swapper/0 (struct addr:ffffffff81e13500)
4.  [    1] systemd (struct addr:ffff88--7c460000)
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/srv/libvmi/libvmi# ./examples/vmi-module-
    list pv-attacker
11. test
12. x86_pkg_temp_thermal
14. Coretemp
15. crct10dif_pclmul
16. ...
```

Figure 5. Running processes and injected modules in the attacker's virtual machine.

provides virtual machine introspection functionality on Xen and KVM, and bridges the semantic gap by reconstructing high-level state information from low-level physical memory data, experiments were performed using LibVMI as an introspection tool to capture evidence related to the two sample attacks. This was accomplished by installing Xen 4.6 in Debian 8 with the privileged Dom0 and configuring the two paravirtualization guests in DomU with Kernel 3.10.100 and Ubuntu 16.04.5, respectively. LibVMI (release 0.12) [12] installed on Dom0 was employed to capture all the running processes and the Linux modules injected in the attacker's guest virtual machine.

Figure 5 shows the running processes and injected modules in the attacker's virtual machine during the CVE-2017-7228 attack. Lines 1 and 10 show that two programs, `vmi-process-list` and `vmi-module-list`, were executed to capture the running processes and modules in the attacker's virtual machine (`pv-attacker`). Lines 3–9 are the captured processes (each line lists the process number, process name and kernel task list address where the process name was retrieved). Lines 11 to 16 provide information about the captured modules; each line shows the module name. Comparisons of the captured processes and modules during the attack against those collected at an earlier time help identify the attack process (`attack`) in Line 9 and the injected attack module (`test`) in Line 11. The module file extension `.ko` is omitted by the program.

While an introspection tool such as LibVMI is effective at detecting hypervisor attacks, it has some limitations. First, in order to access memory consistently, the tool pauses and resumes the guest virtual machine – the experiment revealed that LibVMI paused the attacker's virtual machine for 0.035756 seconds and 0.036173 seconds when capturing the running processes and injected modules, respectively. Second, because virtual machine introspection is only effective during an attack, an attacker could easily utilize an in-virtual-machine timing mechanism (e.g., `kprobes`, a tracing framework built into the kernel) to evade passive virtual machine introspection [29]. Third, storing the captured snapshots of guest VMs for forensic analyses often requires large amounts of storage space.

5. Conclusions

The analysis and classification of recently-reported Xen and KVM vulnerabilities have contributed to the creation of hypervisor attack profiles. The profiles reveal that most attacks on the two hypervisors are due to vulnerabilities arising from the soft memory management unit and the input/output and networking functionalities; the two most common types of hypervisor attacks are denial-of-service and privilege escalation; and most attacks originate from guest virtual machines.

Experiments involving two sample attacks on the Xen and KVM hypervisors provide insights into evidence coverage and the evidence needed to reconstruct attacks during forensic investigations. The most valuable evidence resides in runtime system memory, and obtaining this evidence with guaranteed integrity requires virtual machine introspection techniques that examine the states of guest virtual machines from the hypervisor level while ensuring strong isolation from the guest virtual machines.

Future research will focus on constructing detailed attack paths from the snapshots of attackers' virtual machines, and addressing the timing and memory issues that come into play when using virtual machine introspection.

References

[1] A. Belay, A. Bittau, A. Mashtizadeh, D. Terei, D. Mazieres and C. Kozyrakis, Dune: Safe user-level access to privileged CPU features, *Proceedings of the Tenth USENIX Symposium on Operating Systems Design and Implementation*, pp. 335–348, 2012.

[2] J. Boutoille and G. Campana, Xen Exploitation Part 3: XSA-182, Qubes escape, *Quarkslab's Blog* (`blog.quarkslab.com/xen-exploitation-part-3-xsa-182-qubes-escape.html`), August 4, 2016.

[3] B. Dolan-Gavitt, B. Payne and W. Lee, Leveraging Forensic Tools for Virtual Machine Introspection, School of Computer Science, Georgia Institute of Technology, Atlanta, Georgia, 2011.

[4] 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*, vol. 4(9), pp. 719–727, 2013.

[5] T. Garfinkel and M. Rosenblum, A virtual machine introspection based architecture for intrusion detection, *Proceedings of the Network and Distributed System Security Symposium*, pp. 191–206, 2003.

[6] R. Goldberg, Survey of virtual machine research, *IEEE Computer*, vol. 7(9), pp. 34–45, 1974.

[7] M. Graziano, A. Lanzi and D. Balzarotti, Hypervisor memory forensics, *Proceedings of the Sixteenth International Symposium on Research in Attacks, Intrusions and Defenses*, pp. 21–40, 2013.

[8] J. Horn, Pandavirtualization: Exploiting the Xen Hypervisor, Project Zero, Google, Mountain View, California (`googleprojectzero.blogspot.com/2017/04/pandavirtualization-exploiting-xen.html`), April 7, 2017.

[9] L. Joshi, M. Kumar and R. Bharti, Understanding threats to hypervisor, its forensics mechanism and its research challenges, *International Journal of Computer Applications*, vol. 119(1), pp. 1–5, 2015.

[10] J. Kloster, J. Kristensen and A. Mejlholm, Efficient Memory Sharing in the Xen Virtual Machine Monitor, DAT5 Semester Thesis Report, Department of Computer Science, Aalborg University, Aalborg, Denmark, 2006.

[11] KVM Contributors, Kernel Virtual Machine, KVM (`www.linux-kvm.org/page/Main_Page`), 2019.

[12] LibVMI Community, LibVMI: LibVMI Virtual Machine Introspection, LibVMI (`libvmi.com`), 2019.

[13] C. Liu, A. Singhal and D. Wijesekera, A layered graphical model for cloud forensic mission attack impact analysis, in *Advances in Digital Forensics XIV*, G. Peterson and S. Shenoi (Eds.), Springer, Cham, Switzerland, pp. 263–289, 2018.

[14] S. Lowe, 2015 State of Hyperconverged Infrastructure Market Report, ActualTech Media, Bluffton, South Carolina, 2015.

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

[16] A. Moser, C. Kruegel and E. Kirda, Exploring multiple execution paths for malware analysis, *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 231–245, 2007.

[17] National Institute of Standards and Technology, NIST National Vulnerability Database, Gaithersburg, Maryland (`nvd.nist.gov`), 2019.

[18] B. Pariseau, KVM reignites Type 1 vs. Type 2 hypervisor debate, TechTarget, Newton, Massachusetts (`searchservervirtualization.techtarget.com/news/2240034817/KVM-reignites-Type-1-vs-Type-2-hypervisor-debate`), April 15, 2011.

[19] B. Payne, Simplifying Virtual Machine Introspection Using LibVMI, Sandia Report SAND2012-7818, Sandia National Laboratories, Albuquerque, New Mexico, 2012.

[20] D. Perez-Botero, J. Szefer and R. Lee, Characterizing hypervisor vulnerabilities in cloud computing servers, *Proceedings of the International Workshop on Security in Cloud Computing*, pp. 3–10, 2013.

[21] G. Popek and R. Goldberg, Formal requirements for virtualizable third generation architectures, *Communications of the ACM*, vol. 17(7), pp. 412–421, 1974.

[22] QEMU, QEMU – The FAST! Processor Emulator (`www.qemu.org`), 2019.

[23] 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, *Proceedings of the Workshop on I/O Virtualization*, 2008.

[24] J. Shi, Y. Yang and C. Tang, Hardware assisted hypervisor introspection, *SpringerPlus*, vol. 5(647), 2016.

[25] Y. Song, H. Wang and T. Soyata, Hardware and software aspects of VM-based mobile-cloud offloading, in *Enabling Real-Time Mobile Cloud Computing through Emerging Technologies*, T. Soyata (Ed.), IGI Global, Hershey, Pennsylvania, pp. 247–271, 2015.

[26] J. Szefer, E. Keller, R. Lee and J. Rexford, Eliminating the hypervisor attack surface for a more secure cloud, *Proceedings of the Eighteenth ACM Conference on Computer and Communications Security*, pp. 401–412, 2011.

[27] A. Thongthua and S. Ngamsuriyaroj, Assessment of hypervisor vulnerabilities, *Proceedings of the International Conference on Cloud Computing Research and Innovations*, pp. 71–77, 2016.

[28] R. Uhlig, G. Neiger, D. Rodgers, A. Santoni, F. Martins, A. Anderson, S. Bennett, A. Kagi, F. Leung and L. Smith, Intel virtualization technology, *IEEE Computer*, vol. 38(5), pp. 48–56, 2005.

[29] G. Wang, Z. Estrada, C. Pham, Z. Kalbarczyk and R. Iyer, Hypervisor introspection: A technique for evading passive virtual machine monitoring, *Proceedings of the Ninth USENIX Workshop on Offensive Technologies*, 2015.

[30] Xen Project, x86 Paravirtualized Memory Management (`wiki.xen.org/wiki/X86_Paravirtualised_Memory_Management`), 2019.

[31] Xen Project, Xen Project Software Overview (`wiki.xen.org/wiki/Xen_Project_Software_Overview`), 2019.