Characterizing Hypervisor Vulnerabilities in Cloud Computing Servers

Diego Perez-Botero, Jakub Szefer and Ruby B. Lee
Princeton University, Princeton, NJ, USA
diegop@cs.princeton.edu, {szefer,rblee}@princeton.edu

ABSTRACT

The rise of the Cloud Computing paradigm has led to security concerns, taking into account that resources are shared and mediated by a Hypervisor which may be targeted by rogue guest VMs and remote attackers. In order to better define the threats to which a cloud server's Hypervisor is exposed, we conducted a thorough analysis of the codebase of two popular open-source Hypervisors, Xen and KVM, followed by an extensive study of the vulnerability reports associated with them. Based on our findings, we propose a characterization of Hypervisor Vulnerabilities comprised of three dimensions: the trigger source (i.e. where the attacker is located), the attack vector (i.e. the Hypervisor functionality that enables the security breach), and the attack target (i.e. the runtime domain that is compromised). This can be used to understand potential paths different attacks can take, and which vulnerabilities enable them. Moreover, most common paths can be discovered to learn where the defenses should be focused, or conversely, least common paths can be used to find yet-unexplored ways attackers may use to get into the system.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Reliability, Availability, and Serviceability; D.4.6 [**Operating Systems**]: Security and Protection—*Invasive Software*

Keywords

Hypervisor Vulnerabilities; Vulnerability Categorization; Attack Vectors; Secure Cloud Computing; Virtualization

1. INTRODUCTION

Virtual Machines (VMs) have become commonplace in modern computing, as they enable the execution of multiple isolated Operating System instances on a single physical machine. This increases resource utilization, makes administrative tasks easier, lowers overall power consumption, and enables users to obtain computing resources on demand. Virtualized environments are usually implemented with the use of a Hypervisor, which is a software layer that lies between the Virtual Machines (VMs) and the physical hardware. The Hypervisor allocates resources to the VMs, such as main memory and peripherals. It is in charge of providing each VM with the illusion of being run on its own hardware, which is done by exposing a set of virtual hardware devices (e.g. CPU, Memory, NIC, Storage) whose tasks are then scheduled on the actual physical hardware. These services come at a price: Hypervisors are large pieces of software, with 100,000 lines of code or more. As a result, researchers have been tackling security concerns of traditional Hypervisors, e.g. [22], further motivated by numerous bug reports disclosed for popular Hypervisors (e.g. Xen, KVM, OpenVZ) in a variety of software vulnerability databases, including SecurityFocus [20] and NIST's Vulnerability Database [15].

While the databases provide a plethora of information, not much analysis of the information has thus far been performed. Our work aims to fill this gap through an extensive study of the vulnerability reports associated with Xen and KVM.

The goal of this paper is to characterize the security vulnerabilities of Hypervisors, based on real attacks. Our key contributions are:

1. Three classifications for Hypervisor vulnerabilities based on (1) the Hypervisor functionality where the vulnerability arises, (2) the source that triggers such vulnerability, and (3) the target that is affected by the security breach.
2. An integration of these three classifications to:
   (a) Show potential attack paths (Section 6).
   (b) Understand existing attacks (Section 7.1).
   (c) Help focus defenses (Section 7.2).
   (d) Assist in the discovery of new attacks (Section 7.3).

The rest of the paper is organized as follows. Section 2 provides background on Hypervisors. Section 3 gives a high-level view of Hypervisor vulnerabilities. Sections 4, 5 and 6 describe our extensive analysis and classification of Hypervisor vulnerabilities and potential attack paths. Section 7 describes an existing attack, and some Hypervisor defenses that have been proposed. Section 8 discusses related work. We conclude in Section 9.

2. BACKGROUND ON HYPERVISORS

The virtualization marketplace is comprised of both mature (e.g. VMWare and Xen) and up-and-coming (e.g. KVM and Hyper-V) participants. Of the four main Hypervisor offerings, which take up 93% of the total market share [8],

CloudComputing'13, May 8, 2013, Hangzhou, China.

Diego Perez-Botero, Jakub Szefer and Ruby B. Lee,
"Characterizing Hypervisor Vulnerabilities in Cloud Computing Servers,"
in Proceedings of the Workshop on Security in Cloud Computing (SCC), May 2013.

two are closed-source (VMWare and Hyper-V) and two are open-source (Xen and KVM). Recent surveys [8, 9] suggest that the number of different Hypervisor brands deployed in datacenters is broad and expanding, with a multi-Hypervisor strategy becoming the norm. As such, the percentage of datacenters actively using a specific Hypervisor to host client VMs is known as that Hypervisor's *presence*. Under that definition, VMWare has a total presence of 81%, and 52% of the datacenters use it as their primary Hypervisor, followed by Xen (81% presence, 18% as primary), KVM (58% presence, 9% as primary), and Microsoft's Hyper-V (43% presence, 9% as primary) [8, 9].

We decided not to study VMWare and Hyper-V because of the dearth of public knowledge about their internals. Public Code Vulnerabilities and Exposures (CVEs) for VMWare are always from an outsider's perspective. As such, most of those CVEs focus on network attacks targeting remote management software (e.g. Cross-Site Scripting in CVE-2012-5050). Meanwhile, we were only able to find three CVEs for Hyper-V (i.e. CVE-2011-1872, CVE-2010-3960 and CVE-2010-0026), which does not constitute a representative sample set. Consequently, we have decided to focus on Xen and KVM. Considering Xen's and KVM's influence over the virtualization marketplace, with 81% and 51% datacenter presence respectively, understanding their vulnerabilities can benefit millions of users worldwide.

Below, we briefly summarize Xen's and KVM's architectural traits and their different Hypervisor designs.

2.1 Xen

Xen is a very well-known Open Source Hypervisor, in use since 2003. As shown in Figure 1, Xen is a Type-I (bare metal) Hypervisor, running directly on top of the hardware and managing all of the host's resources. It also has a privileged VM named Dom0, which carries out all of the VM management actions (e.g. start, stop and migrate guest VMs). The Dom0 VM is a full custom-tailored Linux kernel that is aware of the Xen deployment, whereas the normal guest VMs usually run in full virtualization mode (HVM mode), which emulates the entire system (i.e. BIOS, HDD, CPU, NIC) and does not require any modifications to the guest OS. In addition to basic administrative tasks, Dom0 exposes the emulated devices by connecting an instance of a device emulator (i.e. QEMU) to each guest VM.

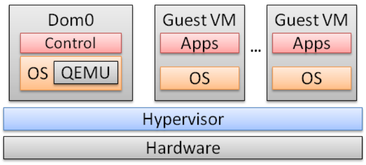

Figure 1: Xen Architecture.

2.2 KVM

KVM is a relatively new open-source project, which dates back to Red Hat's acquisition of Qumranet in 2008. Its adoption has spiked since it was made part of the main Linux kernel branch starting from version 2.6.20, becoming the main virtualization package in Ubuntu, Fedora, and other mainstream Linux operating systems. From Figure 2, one can identify many differences with Xen. Each guest VM runs as a separate user process and has a corresponding QEMU device emulation instance running with it. The Hypervisor itself runs as a module inside a host Operating System, which makes KVM a Type-II (hosted) Hypervisor.

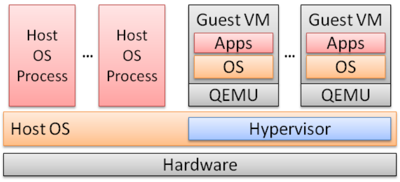

Figure 2: KVM Architecture.

3. OVERVIEW OF VULNERABILITIES

We searched a set of well-known vulnerability databases for reports regarding KVM and Xen: NIST's National Vulnerability Database (NVD) [15], SecurityFocus [20], Red Hat's Bugzilla [17] and CVE Details [3]. Fortunately, all vulnerability reports are assigned a unique CVE Identifier by a CVE Numbering Authority (CNA) and all CNAs use the MITRE Corporation as an intermediary to guarantee the uniqueness of their identifiers, making it easy to eliminate duplicate reports. According to the CVE reports, 59 vulnerabilities have been identified in Xen and 38 in KVM as of July 15, 2012.

Successful exploitation of a vulnerability leads to an attack, which can hinder the Confidentiality, Integrity, or Availability of the Hypervisor or one of its guest VMs. Each CVE report explicitly indicates the type of security breach that it can lead to as a combination of those three security properties. Roughly 50% of vulnerabilities reported so far can lead to security breaches in all three fronts. The second most common effect of exploiting these vulnerabilities is to only pose a threat to the availability of the Hypervisors (Denial of Service).

4. HYPERVISOR FUNCTIONALITIES AS ATTACK VECTORS

To better understand the different vulnerabilities, we considered 11 functionalities that a traditional Hypervisor provides and mapped vulnerabilities to them:

1. Virtual CPUs
2. Symmetric Multiprocessing (SMP)
3. Soft Memory Management Unit (MMU)
4. Interrupt and Timer Mechanisms
5. I/O and Networking
6. Paravirtualized I/O
7. VM Exits
8. Hypercalls
9. VM Management (configure, start, pause and stop VMs)
10. Remote Management Software
11. Hypervisor Add-ons

Categories 1 through 6 present the virtualized hardware infrastructure that VMs require to operate properly. VM Exits and Hypercalls (Categories 7 and 8) are mechanisms through which VMs can delegate sensitive operations to the Hypervisor. Category 9 deals with facilities needed by the Hypervisor to manage VM state. Category 10 deals with non-essential remote management, while Category 11 allows optional add-on modules to the Hypervisor. We further explain these categories below, and describe an actual attack from a CVE report. The CVE reports mentioned throughout this section are readily available online.

Diego Perez-Botero, Jakub Szefer and Ruby B. Lee,
"Characterizing Hypervisor Vulnerabilities in Cloud Computing Servers,"
in Proceedings of the Workshop on Security in Cloud Computing (SCC), May 2013.

① **Virtual CPUs**: A set of virtual CPUs (vCPUs) is assigned to each guest VM being hosted by a Hypervisor. The state of each of these vCPUs is saved to and loaded from their respective VM's Virtual Machine Control Structure (VMCS) guest-state area. Since vCPUs must mirror a physical CPU's actions for each and every machine language instruction, the Hypervisor must handle register states appropriately and schedule vCPU tasks to the physical CPUs while making any necessary translations back and forth.

CVE-2010-4525 is an example of a disclosure of Hypervisor memory contents through vCPU registers because of an incomplete initialization of the vCPU data structures, where one of the padding fields was not zeroed-out. Given that the memory for the data structure is allocated in kernel space, the padding field might end up containing information from data structures previously used by the Hypervisor.

② **Symmetric Multiprocessing (SMP)**: Hypervisors can host guest VMs with SMP capabilities, which leads to the possibility of two or more vCPUs belonging to a single VM being scheduled to the physical CPU cores in parallel. This mode of operation adds complexity to the management of guest VM state and requires additional precautions at the moment of deciding a vCPU's Current Privilege Level (CPL, e.g., Ring 0 or Ring 3).

SMP vulnerabilities arise from Hypervisor code making assumptions that only hold true on single-threaded processes. For example, CVE-2010-0419 refers to a bug that permitted malicious Ring 3 processes to execute privileged instructions when SMP was enabled because of the presence of a race condition scenario. To do so, they would invoke a legitimate I/O instruction on one thread and attempt to replace it with a privileged one from another thread right after KVM had checked its validity, but before it was executed.

③ **Soft MMU**: Guest VMs cannot be granted direct access to the MMU, as that would allow them to access memory belonging to the Hypervisor and other co-hosted VMs. Under the absence of a virtualization-aware hardware MMU, such as Extended Page Tables (EPT), a Soft MMU is run by the Hypervisor to maintain a shadow page table for each guest VM. Every page mapping modification invoked by a VM is intercepted by the Soft MMU so as to adjust the shadow page tables accordingly.

Vulnerabilities in the Soft MMU's implementation are dangerous because they may lead to the disclosure of data in arbitrary address spaces, such as a co-hosted guest VM's memory segment or the Hypervisor's memory segment. In the specific case of CVE-2010-0298, KVM's emulator always uses Ring 0 privilege level when accessing a guest VM's memory on behalf of the guest VM's code. Given that MMIO instructions are emulated, an unprivileged (Ring 3) application running inside a VM could leverage access to an MMIO region (e.g. framebuffer) to trick KVM into executing a malicious instruction that modifies that same VM's kernel-space memory.

④ **Interrupt and Timer Mechanisms**: A Hypervisor must emulate the interrupt and timer mechanisms that the motherboard provides to a physical machine. These include the Programmable Interval Timer (PIT), the Advanced Programmable Interrupt Controller (APIC), and the Interrupt Request (IRQ) mechanisms.

In the case of CVE-2010-0309, lack of validation of the data contained in the PIT-related data structures enabled a rogue VM to cause a full host OS crash, a serious denial-of-service attack.

⑤ **I/O and Networking**: The Hypervisor also emulates I/O and networking. Xen and KVM make device emulation possible through division of labor, by having two types of device drivers. Front-end drivers reside inside the guest VMs and run in Ring 0, providing the usual abstraction that the guest OS expects. Nonetheless, those drivers cannot access physical hardware directly, given that the Hypervisor must mediate user accesses to shared resources. Therefore, front-end drivers communicate with back-end drivers, which have full access to the underlying hardware, in order to fulfill the requested operations. In turn, back-end drivers enforce access policies and multiplex the actual devices. KVM and Xen employ QEMU's back-end drivers by default.

Device emulation is usually implemented in higher-level languages (e.g. C and C++), so the data abstractions are richer but more dangerous when hijacked. Very elaborate attacks are enabled by the expressiveness of higher-level languages like C. For example, CVE-2011-1751 describes a bug that was used to develop the Virtunoid attack [5]. QEMU tried to hot-unplug whichever device the programmers desired, regardless of the device's support for hot-unplugging. Therefore, the lack of state cleanup by some virtual devices resulted in *use-after-free* opportunities, where data structures that were previously being used by a hot-unplugged virtual device remained in memory and could be hijacked with executable code by an attacker.

⑥ **Paravirtualized I/O**: paravirtualized VMs run modified guest kernels that are virtualization-aware and use special hypercall APIs to interact with the Hypervisor directly. Paravirtualization of I/O operations decreases the number of transitions between the guest VM and the Hypervisor, resulting in performance gains. This scenario requires special front-end and back-end drivers which are not necessarily developed by the same vendor as the one responsible for regular device emulation (e.g. QEMU).

Paravirtualized I/O vulnerabilities and emulated I/O vulnerabilities are very much alike. They are rooted in the interactions between front-end and back-end drivers, as well as those between back-end drivers and the outside world. For instance, CVE-2008-1943 describes a vulnerability in Xen that allowed paravirtualized front-end drivers to cause denial-of-service conditions and possibly execute arbitrary code with Dom0 privileges. This could be done by sending a malicious shared framebuffer descriptor to trick Xen into allocating an arbitrarily large internal buffer inside Dom0.

⑦ **VM Exits** are the mechanism used by the Hypervisor to intercept and carry out operations invoked by guest VMs that require Virtual Machine eXtensions (VMX) root privileges. These VM-to-Hypervisor interfaces are architecture-dependent (e.g. different code for x86 than for AMD64) and are very well specified in the architecture manuals. They are usually implemented using low-level programming languages (Assembly or Machine language), relying on restrictive bitwise operations. For Intel VT-x, this code is the one supporting all operations described in chapters 23 through 33 of Intel's Software Developer's Manual [10].

The fact that VM Exit-handling code does not possess very rich data structures means that vulnerabilities hardly have any exploitable effects other than a host or guest VM crash (Denial-of-Service). For example, all VMCS fields have a unique 32-bit field-encoding, which rules out common vulnerabilities that arise from variable-size input, such as buffer overflows. According to CVE-2010-2938, requesting a full VMCS dump of a guest VM would cause the entire host to crash when running Xen on a CPU without Extended Page Table (EPT) functionality. The reason for this was that Xen would try to access EPT-related VMCS fields without first verifying hardware support for those fields, allowing privileged (Ring 0) guest VM applications to trigger a full denial-of-service attack on certain hosts at any time.

⑧ **Hypercalls** are analogous to system calls in the OS world. While VM Exits are architecture-specific (e.g. AMD64, x86), hypercalls are Hypervisor-specific (e.g. Xen, KVM) and provide a procedural interface through which guest VMs can request privileged actions from the Hypervisor. Hypercalls can be used to query CPU activity, manage Hard Disk partitions, and create virtual interrupts.

Hypercall vulnerabilities can present an attacker, who controls a guest VM, with a way to attain escalated privileges over the host system's resources. Case in point, CVE-2009-3290 mentions the fact that KVM used to allow unprivileged (Ring 3) guest callers to issue MMU hypercalls. Since the MMU command structures must be passed as an argument to those hypercalls by their physical address, they only make sense when issued by a Ring 0 process. Having no access to the physical address space, the Ring 3 callers could still pass random addresses as arguments to the MMU hypercalls, which would either crash the guest VM or, in the worst case, read or write to kernel-space memory segments.

⑨ **VM Management** functionalities make up the set of basic administrative operations that a Hypervisor must support. The configuration of guest VMs is expressed in terms of their assigned virtual devices, dedicated PCI devices, main memory quotas, virtual CPU topologies and priorities, etc. The Hypervisor must then be able to start, pause and stop VMs that are true to the configurations declared by the cloud provider. These tasks are initiated by Xen's Dom0 and KVM's libvirt toolkit [14].

Kernel images must be decompressed into memory and interpreted by the management domain when booting up a VM. CVE-2007-4993 indicates that Xen's bootloader for paravirtualized images used Python exec() statements to process the custom kernel's user-defined configuration file, leading to the possibility of executing arbitrary python code inside Dom0. By changing the configuration file to include the line shown in Listing 1, a malicious user could trick Dom0 into issuing a command that would trigger the destruction of another co-hosted domain (substituting *id* with the victim domain's ID).

```
default "+str(os.system("xm destroy id"))+"
```

Listing 1: Contents of /boot/grub/grub.conf for an attack on Dom0 with a user-provided kernel

⑩ **Remote Management Software**: These pieces of software are usually web applications running as a background process and are not essential for the correct execution of the virtualized environment. Their purpose is generally to facilitate the Hypervisor's administration through user-friendly web interfaces and network-facing virtual consoles.

Vulnerabilities in these bundled applications can be exploited from anywhere and can lead to full control over the virtualized environment. For example, CVE-2008-3253 describes a Cross-Site Scripting attack on a remote administration console that exposed all of Xen's VM management actions to a remote attacker after stealing a victim's authentication cookies.

⑪ **Hypervisor Add-ons**: Hypervisors like Xen and KVM have modular designs that enable extensions to their basic functionalities – Hypervisor Add-ons. For example, the National Security Agency (NSA) has developed their own version of Xen's Security Modules (XSM) called FLASK.

Hypervisor add-ons increase the likelihood of Hypervisor vulnerabilities being present, since they increase the size of the Hypervisor's codebase. For example, CVE-2008-3687 describes a heap overflow opportunity in one of Xen's optional security modules, FLASK, which results in an escape from an unprivileged domain directly to the Hypervisor.

4.1 Breakdown of Vulnerabilities

We analyzed all of KVM's and Xen's CVE reports from the 4 vulnerability databases, labeling each with its functionality-based attack vector. Our resulting vulnerability breakdowns are presented in Table 1. It can be observed that the Device Emulation categories (i.e. I/O and Networking along with Paravirtualized I/O) account for more than one third of the known vulnerabilities for each of the Hypervisors (33.9% of Xen's and 39.5% of KVM's vulnerabilities). This can be attributed to the variety of back-end drivers that are supported by both Xen and KVM. A normal QEMU installation is capable of emulating all sorts of virtual devices (e.g. NIC, display, audio) and different models of each type (Intel Ethernet i82559C, Realtek Ethernet rtl8139, etc.), leading to a considerable number of distinct use cases and a fairly large codebase.

Table 1: Breakdown of known vulnerabilities under functionality-based classification for Xen and KVM.

Attack Vector	Xen	KVM
Virtual CPUs	5 (8.5%)	8 (21.1%)
SMP	1 (1.7%)	3 (7.9%)
Soft MMU	4 (6.8%)	2 (5.3%)
Interrupt and Timer Mechanisms	2 (3.4%)	4 (10.5%)
I/O and Networking	11 (18.6%)	10 (26.3%)
Paravirtualized I/O	9 (15.3%)	5 (13.2%)
VM Exits	4 (6.8%)	2 (5.3%)
Hypercalls	2 (3.4%)	1 (2.6%)
VM Management	7 (11.9%)	2 (5.3%)
Remote Management Software	9 (15.3%)	1 (2.6%)
Hypervisor add-ons	5 (8.5%)	0 (0.0%)
Total	**59**	**38**

The number of Remote Management Software vulnerabilities in Xen (accounting for 15.3% of its vulnerabilities) shows that non-essential services may increase the attack surface significantly. More interestingly, KVM reports a markedly lower contribution from VM Management vulnerabilities towards the total (5.3% in KVM vs 11.9% in Xen). This might suggest that KVM's architectural decision of running the libvirt toolkit (in charge of VM Management functionalities) as an additional module inside Hypervisor space is more secure than Xen's decision of allocating an entire privileged VM (Dom0) for the same purpose. After all, Xen's Dom0 domain is a specialized linux kernel, which means that it needs to execute at least a minimal set of OS services in order to run, therefore increasing the likelihood of bugs.

5. FURTHER CHARACTERIZATION OF HYPERVISOR VULNERABILITIES

Our analysis of KVM and Xen vulnerability reports gave rise to two additional complementary classifications: trigger source and attack target. A Hypervisor vulnerability manifests itself inside a module's code, but can be triggered from a variety of runtime spaces and can target one or more of those runtime spaces. Listed from lowest to highest privilege level: (1) Network, (2) Guest VM's User-Space, (3) Guest VM's Kernel-Space, (4) Dom0/Host OS, (5) Hypervisor.

The trigger source and attack target are of great importance when assessing a vulnerability's ease of exploitability and impact, respectively. The trigger source can be determined by comparing the restrictions of each of the runtime spaces with the execution rights required to reproduce the vulnerability. Since these five categories correspond to hierarchical privilege levels, we show the least possible privilege level for the trigger source, and the greatest possible privilege level for the attack target in Tables 2 and 3.

5.1 Trigger Sources and Attack Targets

① **Network**: This is the least privileged runtime space, but also the easiest to attain. Any remote user can initiate an attack on a Hypervisor and its guest VMs if it is located in a subnet from which the machine running the Hypervisor is reachable.

② **Guest VM's User-Space**: Almost any code can be executed from a guest VM's Ring 3; however, some functionality will be limited by the OS or the Hypervisor (causing an exception). Nevertheless, it is easiest to get user-space code to run, so any exploits from this ring are attractive to an attacker. For example, CVE-2010-4525 mentions an attack from a guest VM's Ring 3 involving the CPUID x86 instruction.

③ **Guest VM's Kernel-Space**: Injecting malicious OS-level (Ring 0) code requires compromising the OS security. Interestingly, in IaaS cloud deployments, tenants can simply lease VMs and run their OS of choice – one which may already be malicious. For example, CVE-2008-1943 mentions an attack from a Guest VM's Kernel-Space, as it requires control over the paravirtualized front-end driver.

④ **Dom0/Host OS**: Some runtime spaces have privilege levels that lie between those of a guest VM's OS and the ones possessed by the Hypervisor. In Xen's case, Dom0 is a privileged VM with direct access to I/O and networking devices. At the same time, Dom0 is allowed to invoke VM Management operations. While KVM does not have a Dom0 equivalent, the fact that the Hypervisor is part of a fully-operational Linux kernel gives way to other types of threats (e.g. local users in the host system).

⑤ **Hypervisor**: This is the most desired runtime space because it has Ring -1 privileges, so any command can be run from this space. The Hypervisor can access any resource in the host system (i.e. memory, peripherals, CPU state, etc), which means that it can access every guest VM's resources.

5.2 Breakdown of Vulnerabilities

As can be observed in Table 2, the most common trigger source is Guest VM User-Space (Ring 3), accounting for 39.0% of Xen's and 34.2% of KVM's vulnerabilities. This is worrying, as it indicates that any unprivileged guest VM user has the necessary privileges to pose a threat to the underlying Hypervisor. The Guest VM Kernel-Space is the second most common trigger source, with roughly 32% of the total in both cases. Hence, 71.2% of all Xen and 65.8% of all KVM vulnerabilities are triggered from a guest VM. Also note that there are no vulnerabilities with Hypervisor space as their trigger source, which makes sense because an attacker who has control over the Hypervisor already has the maximum privilege level attainable.

Table 2: Breakdown of known vulnerabilities under trigger source classification for Xen and KVM.

Trigger Source	Xen	KVM
Network	11 (18.6%)	2 (5.3%)
Guest VM User-Space	23 (39.0%)	13 (34.2%)
Guest VM Kernel-Space	19 (32.2%)	12 (31.6%)
Dom0/Host OS	6 (10.2%)	11 (28.9%)
Hypervisor	0 (0.0%)	0 (0.0%)
Total	**59**	**38**

Two differences between the two Hypervisors stand out: Xen is much more vulnerable to network-based attacks than KVM, but KVM is more sensitive to Host OS-based attacks. The first observation follows from our attack vector analysis (Section 4), which showed that Remote Management Software vulnerabilities are a big problem for Xen. On the other hand, KVM's sensitivity to Host OS threats is to be expected because, being part of the main Linux kernel branch, its code can be invoked by other kernel-space processes running on the host, leaving it exposed to malicious privileged local users.

Table 3: Breakdown of known vulnerabilities under target-based classification for Xen and KVM.

Attack Target	Xen	KVM
Network	0 (0.0%)	0 (0.0%)
Guest VM User-Space	0 (0.0%)	0 (0.0%)
Guest VM Kernel-Space	12 (20.3%)	9 (23.7%)
Dom0/Host OS	25 (42.4%)	11 (28.9%)
Hypervisor	22 (37.3%)	18 (47.4%)
Total	**59**	**38**

It can be observed from Table 3 that Dom0 is a more common target than the Hypervisor in Xen, whereas KVM shows the opposite behaviour (its Host OS is less common than the Hypervisor as a target). This difference between the two Hypervisors is due to the location of the Device Emulation back-end drivers, which are found in Dom0 with Xen and in the Hypervisor with KVM. The Device Emulation functionalities contribute more than one third of the known vulnerabilities in both Hypervisors, so the location of the back-end drivers has great influence over the relative distribution of vulnerabilities among the possible attack targets.

6. HYPERVISOR ATTACK PATHS

Tables 4 and 5 show an integration of all of our three attack classifications for Xen and KVM, respectively. Each row illustrates a potential attack path; starting at some trigger source, exploiting a specific Hypervisor functionality, to attack a set of targets. In each row, the trigger sources are less privileged software entities, while the attack targets are the more privileged software entities, thus enabling privilege escalation. A co-located hostile VM can take multiple iterations through the attack paths (left to right, wraparound to left to right, etc.) to achieve privilege escalation, eventually attaining Dom0/Host OS or Hypervisor-level privileges. When the attacker achieves these elevated privileges, it can see, modify or deny services to a victim VM, thus breaching the victim's confidentiality, integrity or availability.

Table 4: Xen's Vulnerability Map in tabular form

Trigger Source				Attack Vector	Attack Target		
NW	Usr	OS	Dom0		OS	Dom0	HV
	X		X	Virtual CPUs	X		X
	X			SMP	X		
	X	X	⊗	Soft MMU	X		⊗
	X	X		I&T. Mech.			X
X	X	X		I/O and NW	X	X	
	X	X		Paravirt. I/O	X	X	
	X	X		VM Exits	X		X
		X		Hypercalls			X
	X	⊗	X	VM Management		⊗	X
X				Rem. Mgmt. SW			X
X		X	X	HV add-ons		X	X

Table 5: KVM's Vulnerability Map in tabular form

Trigger Source				Attack Vector	Attack Target		
NW	Usr	OS	Host		OS	Host	HV
	X	X	X	Virtual CPUs	X	X	X
	X			SMP	X	X	
	X	X		Soft MMU	X		X
	X	X	X	I&T. Mech.		X	X
X	X	X	X	I/O and NW	X	X	X
X		X		Paravirt. I/O	X	X	X
	X			VM Exits	X	X	
	X			Hypercalls	X		
			X	VM Management			X
X				Rem. Mgmt. SW			X
				HV add-ons			

Legend: NW = Network; Usr = Guest VM User-Space; OS = Guest VM Kernel-Space; Host = Host OS; HV = Hypervisor; I&T Mech. = Interrupt and Timer Mechanisms; Paravirt. I/O = Paravirtualized I/O; Rem. Mgmt. SW = Remote Management Software

The specially marked ⊗s in Table 4 are an example of a possible 2-step privilege escalation path that a privileged guest VM user (Ring 0) could follow in order to reach Hypervisor runtime space (Ring -1) in a Xen deployment. The first step would be to exploit a *VM Management* vulnerability to gain control of Dom0. A viable attack to achieve this transition is CVE-2007-4993 (see Section 4), which enables the execution of arbitrary python code inside Dom0. Once in control of Dom0, exploiting a *Soft MMU* vulnerability could grant the malicious user control over the most desired runtime space: Ring -1. The Q35 attack (CVE-2008-7096), covered in the next section, could be used to that end.

7. CASE STUDY AND DEFENSES

The goal of our exploration of the vulnerabilities, their classification and creation of the vulnerability maps is to help researchers better understand attacks on Hypervisors and determine where defenses should be concentrated. We now first show how a real world attack can be analyzed using the maps.

7.1 Understanding Existing Attacks

For our case study, we present the Dom0 Attack on Xen from Black Hat USA 2008 [19].

7.1.1 *Case Study: Dom0 Attack on Xen (Black Hat USA 2008)*

This attack [19] revolves around Intel's Q35 chipset for the Core 2 Duo/Quad platforms. In Q35 chipsets, the processor provides the capability to re-claim the physical memory overlapped by the Memory Mapped I/O logical address space. Under normal operation, the REMAPBASE and REMAPLIMIT registers are calculated and loaded by the BIOS. The amount of memory remapped is the range between Top of Low Usable DRAM (TOLUD) and 4 GB. This physical memory will be mapped to the logical address range defined between the REMAPBASE and the REMAPLIMIT registers. The end result is shown in Figure 3.

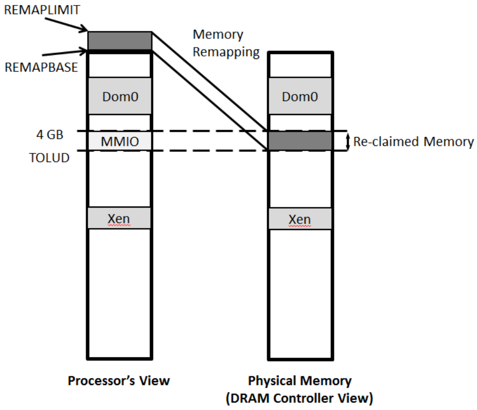

Figure 3: Memory remapping during normal operation of Q35 chipset.

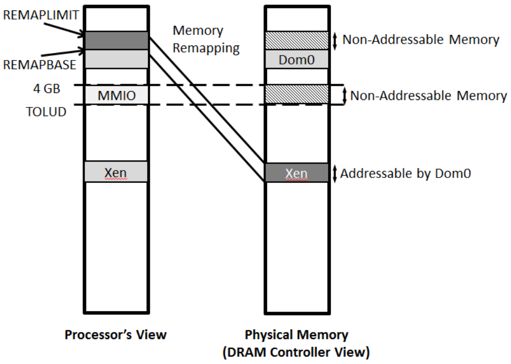

Figure 4: Memory remapping during attack on Q35 chipset.

The Invisible Things team [19] managed to hijack Xen's Hypervisor memory from the Dom0 domain in Q35 chipsets by exploiting the host system's remapping registers. Xen's Dom0 has write access to the host system's REMAPBASE and REMAPLIMIT registers, so a malicious Dom0 kernel is able to set up a memory remapping range pointing to the

Hypervisor's physical memory, as shown in Figure 4. As a result, the Dom0 attacker can read and modify the Hypervisor's memory space during runtime. This constitutes a serious confidentiality and integrity breach, making it possible for the attacker to run any series of instructions with Ring -1 privilege level.

Note that the Invisible Things team [19] employed an already-compromised Dom0 kernel to carry out the Q35 attack. Therefore, in terms of the Xen vulnerability map shown in Table 4, the initial trigger source for the original Q35 attack is Dom0 (which runs in Ring 0). The attack vector is via part of the Soft MMU because memory management and memory control is mediated incorrectly. A mechanism to lock the remapping registers after the BIOS has set them up is a possible defense, eliminating the possibility for a rogue Dom0 domain to write new values to them. Finally, the attack target is the Hypervisor (Ring -1), whose memory space is completely compromised.

7.2 Helping Focus Defenses

While we believe this to be the first categorization of Hypervisor vulnerabilities, researchers have been aware of various attacks on Hypervisors and have proposed a number of defenses. We summarize some of these, then suggest areas to focus defenses based on our Hypervisor vulnerability maps.

One defense strategy against attacks is to make the Hypervisor codebase more resilient to attacks. Projects such as HyperSafe [25] have looked at hardening the code to make it more difficult to inject code and subvert the control flow of the Hypervisor through clever programming techniques. This aims to address attack paths targeting the Hypervisor; however, this does not mean that all attack vectors with the Hypervisor as the attack target are mitigated.

Protecting the Hypervisor kernel from an untrusted management OS [13] is another approach that has been proposed. Such work covers Dom0/Host OS trigger sources, and especially paths with a VM Management attack vector.

Another defense strategy is to use hardware-assisted techniques for protecting the software integrity of the Hypervisor to detect the attacks before they can do damage. For example, Copilot [16] employs a special purpose PCI device to read the physical memory of the target system. HyperCheck [24] looks at using features of the microprocessor, namely the system management mode (SMM) to inspect the Hypervisor. HyperSentry [1] also used the SMM to bypass the Hypervisor for integrity measurement purposes. Such work aims to cover paths toward the Hypervisor attack target.

A fourth defense strategy is removing the Hypervisor altogether. The NoHype [11, 22] architecture for cloud computing eliminates the Hypervisor layer and places VMs directly on top of the physical hardware while still being able to start, stop and run multiple VMs at the same time.

From our analysis of CVEs, we believe that defenses for commodity Hypervisors should start by focusing on Hypervisor correctness. Thorough input validation, proper tracking of context changes, complete initialization of control structures, complete clearing of sensitive data on process termination, and full awareness of the underlying hardware's capabilities would immediately reduce the Hypervisor's attack surface. The emulation of I/O and networking devices proves to be a common point of failure, so Hypervisor vendors should aim at developing a small set of secure back-end drivers instead of trying to provide a large number of virtual devices with overlapping functionality (e.g. e1000, ne2k_pci and rtl8139 networking cards) that are hard to maintain.

7.3 Assisting in the Discovery of New Attacks

While many proposed defenses exist, numerous paths through the Hypervisor vulnerability maps (Table 4 or 5) are not yet covered. However, it should be noted that some of them can be dismissed. For example, it is inconceivable for a remote (network-bound) attacker to directly exploit VM Exit-related vulnerabilities because VM Exits are a mechanism that only exists in the boundary between a VM and the Hypervisor, so the attacker must first gain access to a VM. Even though some attack paths can be ruled out, most of them are valid. The absence of current attacks with a specific [source, vector, target] combination does not necessarily rule out the possibility of a future attack leading to those conditions. For instance, KVM's vulnerability map (Table 5) does not report any existing attacks with a VM's Kernel-Space as the source and the Hypervisor as a target using Hypercalls as an attack vector (i.e. [OS, Hypercalls, HV]), which we know for a fact to be a possibility judging from Xen's vulnerability map (Table 4).

Our vulnerability maps provide a way to assess the coverage of each protection mechanism that a cloud provider can employ. Equally important, they provide a way of identifying weak spots before an actual attack surfaces. If a given [source, vector, target] combination is not addressed by a secure Hypervisor, a malicious user will be able to decide where to concentrate his efforts. Conversely, our work suggests specific areas where the cloud provider can focus hardening efforts to minimize the risk of such attacks.

8. RELATED WORK

To the best of our knowledge, there has been no detailed categorization of Hypervisor vulnerabilities as presented in this paper. Many researchers have looked at security issues in cloud computing and produced surveys of those issues. The surveys (e.g. [23], [26]) focus on various threats for the cloud environment as a whole: abuse of cloud computing resources, insecure APIs, etc. There has also been work on classification of threats based on the different service delivery models of cloud computing [21]. Other works have presented classifications of security issues at different levels, such as network, host or application [2].

As one of its contributions, our work aims to categorize different attack vectors. Outside of cloud computing, researchers have explored categorizing kernel-level rootkits to aid future detection [12]. Others have looked at attack surfaces in cloud computing, however, at the level of user, services and cloud without diving into details of the attack surfaces on the virtualization layer itself [7]. Attack surface inflation [6] has been explored including the change of the attack surface as new components are integrated into an existing system, e.g. adding virtualization. Researchers have also looked at the classification of threats and challenges faced by different components of an IaaS (infrastructure-as-a-service) cloud computing deployment, components such as cloud software, platform virtualization, network connectivity, etc. [4]. Different from all these works, our work focuses on the Hypervisor attack surface.

Interesting work on mapping cloud infrastructure [18] has given insights on how to find a specific target cloud server to attack. There have not been, however, other works which aim, as we do, to map the cloud infrastructure attack paths.

9. CONCLUSIONS

To successfully compromise a system, malicious users must characterize the attack surface available to them and evaluate their possible targets while considering the restrictions of their vantage point (trigger source). In this work, we conducted a thorough analysis of the codebase of two popular Hypervisors, Xen and KVM, followed by an extensive study of vulnerability reports associated with them. Based on our findings, we are the first to propose and integrate three Hypervisor Vulnerability classifications: by Hypervisor functionality, by trigger source, and by attack target. Our integration of these three classifications gives a clear picture of the different Hypervisor modules and runtime spaces that are traversed during the course of a successful attack. We demonstrated the practicality of this abstract model for vulnerability analysis in describing the flow of events involved in a well-known attack that achieved privilege escalation in virtualized systems. By clearly exposing potential attack paths, our Hypervisor vulnerabilities characterization is also actionable: it enables us to see what vulnerabilities have been covered by proposed solutions in past work, and what needs to be covered by new defenses. Our work can assist in better establishing a specific user's security needs and determining the scope of the solutions that might be proposed to address them.

10. REFERENCES

[1] A. M. Azab, P. Ning, Z. Wang, X. Jiang, X. Zhang, and N. C. Skalsky. Hypersentry: enabling stealthy in-context measurement of hypervisor integrity. In *Proceedings of the ACM Conference on Computer and Communications Security*, CCS, pages 38 – 49, October 2010.

[2] R. Bhadauria, R. Chaki, N. Chaki, and S. Sanyal. A survey on security issues in cloud computing. *arXiv, http://arxiv.org/abs/1109.5388*, September 2011.

[3] Cve security vulnerability database. `http://www.cvedetails.com/`.

[4] W. Dawoud, I. Takouna, and C. Meinel. Infrastructure as a service security: Challenges and solutions. In *Proceedings of the International Conference on Informatics and Systems*, INFOS, pages 1 – 8, March 2010.

[5] N. Elhage. Virtunoid: Breaking out of KVM. `nelhage.com/talks/kvm-defcon-2011.pdf`, August 2011.

[6] D. Geer. Attack surface inflation. *IEEE Security Privacy Magazine*, 9(4):85 – 86, July – August 2011.

[7] N. Gruschka and M. Jensen. Attack surfaces: A taxonomy for attacks on cloud services. In *Proceedings of the IEEE International Conference on Cloud Computing*, CLOUD, pages 276 – 279, July 2010.

[8] Nexenta Hypervisor Survey. `http://www.nexenta.com/corp/nexenta-hypervisor-survey`.

[9] Is the Hypervisor Market Expanding or Contracting? `http://www.aberdeen.com/Aberdeen-Library/8157/AI-hypervisor-server-virtualization.aspx`.

[10] Intel. Intel 64 and IA-32 Architectures Software Developer's Manual., October 2011. `http://www.intel.com/content/dam/doc/manual/64-ia-32-architectures-software-developer-manual-325462.pdf`.

[11] E. Keller, J. Szefer, J. Rexford, and R. B. Lee. Nohype: virtualized cloud infrastructure without the virtualization. In *Proceedings of the Annual International Symposium on Computer Architecture*, ISCA, pages 350 – 361, June 2010.

[12] J. Levine, J. Grizzard, and H. Owen. Detecting and categorizing kernel-level rootkits to aid future detection. *IEEE Security Privacy Magazine*, 4(1):24 – 32, January – February 2006.

[13] C. Li, A. Raghunathan, and N. K. Jha. Secure Virtual Machine Execution under an Untrusted Management OS. In *Proceedings of the Conference on Cloud Computing*, CLOUD, pages 172 – 179, July 2010.

[14] libvirt. `http://libvirt.org/`.

[15] National vulnerability database. `http://web.nvd.nist.gov/view/vuln/search`.

[16] N. L. Petroni, Jr., T. Fraser, J. Molina, and W. A. Arbaugh. Copilot - a coprocessor-based kernel runtime integrity monitor. In *Proceedings of the USENIX Security Symposium*, pages 179 – 194, August 2004.

[17] Red hat bugzilla. `https://bugzilla.redhat.com/`.

[18] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proceedings of the ACM Conference on Computer and Communications Security*, CCS, pages 199 – 212, November 2009.

[19] J. Rutkowska and R. Wojtczuk. Preventing and detecting xen hypervisor subversions. `invisiblethingslab.com/resources/bh08/part2-full.pdf`, July 2008.

[20] Securityfocus. `http://www.securityfocus.com/`.

[21] S. Subashini and V. Kavitha. A survey on security issues in service delivery models of cloud computing. *Journal of Network and Computer Applications*, 34(1):1 – 11, 2011.

[22] J. Szefer, E. Keller, R. B. Lee, and J. Rexford. Eliminating the hypervisor attack surface for a more secure cloud. In *Proceedings of the Conference on Computer and Communications Security*, CCS, October 2011.

[23] L. Vaquero, L. Rodero-Merino, and D. Morán. Locking the sky: a survey on iaas cloud security. *Computing*, 91:93 – 118, 2011.

[24] J. Wang, A. Stavrou, and A. Ghosh. Hypercheck: A hardware-assisted integrity monitor. In *Recent Advances in Intrusion Detection*, volume 6307 of *Lecture Notes in Computer Science*, pages 158 – 177. 2010.

[25] Z. Wang and X. Jiang. Hypersafe: A lightweight approach to provide lifetime hypervisor control-flow integrity. In *Proceedings of the IEEE Symposium on Security and Privacy*, S&P, pages 380 – 395, May 2010.

[26] M. Zhou, R. Zhang, W. Xie, W. Qian, and A. Zhou. Security and privacy in cloud computing: A survey. In *Proceedings of the International Conference on Semantics Knowledge and Grid*, SKG, pages 105 –112, November 2010.

Diego Perez-Botero, Jakub Szefer and Ruby B. Lee,
"Characterizing Hypervisor Vulnerabilities in Cloud Computing Servers,"
in Proceedings of the Workshop on Security in Cloud Computing (SCC), May 2013.