CloudVisor: Retrofitting Protection of Virtual Machines in Multi-tenant Cloud with Nested Virtualization

Fengzhe Zhang, Jin Chen, Haibo Chen and Binyu Zang
Parallel Processing Institute
Fudan University
{fzzhang, chenjin, hbchen, byzang}@fudan.edu.cn

ABSTRACT

Multi-tenant cloud, which usually leases resources in the form of virtual machines, has been commercially available for years. Unfortunately, with the adoption of commodity virtualized infrastructures, software stacks in typical multi-tenant clouds are non-trivially large and complex, and thus are prone to compromise or abuse from adversaries including the cloud operators, which may lead to leakage of security-sensitive data.

In this paper, we propose a transparent, backward-compatible approach that protects the privacy and integrity of customers' virtual machines on commodity virtualized infrastructures, even facing a total compromise of the virtual machine monitor (VMM) and the management VM. The key of our approach is the separation of the resource management from security protection in the virtualization layer. A tiny security monitor is introduced underneath the commodity VMM using nested virtualization and provides protection to the hosted VMs. As a result, our approach allows virtualization software (e.g., VMM, management VM and tools) to handle complex tasks of managing leased VMs for the cloud, without breaking security of users' data inside the VMs.

We have implemented a prototype by leveraging commercially-available hardware support for virtualization. The prototype system, called CloudVisor, comprises only 5.5K LOCs and supports the Xen VMM with multiple Linux and Windows as the guest OSes. Performance evaluation shows that CloudVisor incurs moderate slowdown for I/O intensive applications and very small slowdown for other applications.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection

General Terms

Design, Security, Performance

Keywords

Multi-tenant Cloud, Virtual Machine Security, Nested Virtualization

1. INTRODUCTION

Multi-tenant cloud has advantages of providing elastic and scalable computing resources and freeing users from the cumbersome tasks such as configuring, managing and maintaining IT resources. For example, Amazon's Elastic Compute Cloud (EC2) [6] platform provides flexible and resizable computing resources in the form of Xen-based VMs for a number of usage scenarios, including application hosting, content delivering, e-commerce and web hosting [6].

However, multi-tenant cloud also redefines the threat model of computing and raises new security challenges: the security of customers' sensitive data will be a key concern if being put into a third-party multi-tenant cloud. Unfortunately, current multi-tenant cloud platforms adopting commodity virtualization infrastructures usually provide limited assurance for the security of tenants' sensitive data. Many cloud providers only provide "security on your own" guarantee to users' content [8].

There are mainly two reasons for the poor security guarantee provided in current clouds. First, many cloud platforms usually adopt off-the-shelf virtualized infrastructures for the purpose of easing deployment and lowering costs. However, this also introduces the probability of security compromises of leased VMs from the virtualization stack. This is because, the trusted computing base (TCB) for commodity virtualized infrastructures, which includes *both the Virtual Machine Monitor (VMM) and the management VM*, is in the scale of several millions LOCs. Thus, the stack is prone to intrusions and "jail-breaks". For example, by December 2010, there have been 35 and 32 reported vulnerabilities from CVE [2] for VMware and Xen respectively.

Second, tenants from competitive companies or even the cloud operators themselves may be potential adversaries, which might stealthily make unauthorized access to unencrypted sensitive data. For example, a report assessing security risks of cloud computing from Gartner states that, one biggest challenge of cloud computing is "invisibly access unencrypted data in its facility" [26]. Google also recently fired two employees for breaching user privacy [63].

To ameliorate this problem, previous efforts have attempted to completely remove the virtualization layer [31], building a new micro-kernel like VMM [60], or protecting a VMM's control-flow integrity [68]. However, these approaches mostly only protect VMMs from attacks from a malicious guest VM, without consideration of preventing an operator with control of management tools and control VM from tampering with or stealing users' confidential data, especially external storage such as virtual disks. Further, they require changes to the core parts of a VMM [68] or even a complete reconstruction of VMMs [31, 60], thus may pose a notable barrier for adoption in commercially-successful virtualized cloud.

In this paper, we propose an alternative approach that protects leased virtual machines in a multi-tenant cloud. Our approach uses the concept of *nested virtualization* [23, 13] and introduces a tiny security monitor called CloudVisor underneath a mostly unmodified commodity VMM. Unlike previous approaches that incorporate nested virtualization functionality into a commodity VMM [13] for the purpose of multiple-level virtualization, CloudVisor decouples the functionality of nested virtualization from a commodity VMM and makes itself very lightweight in only supporting one VMM.

CloudVisor is responsible for protecting privacy and integrity of resources owned by VMs, while the VMM is still in charge of allocating and managing resources for VMs atop. Such a separation between security protection and resource management allows CloudVisor and the VMM to be independently designed, verified and evolved. As the essential protection logic for VM resources is quite fixed, CloudVisor can be small enough to verify its security properties (e.g., using formal verification methods [34]).

CloudVisor interposes interactions between a VMM and its guest VMs for privacy protection and integrity checking. To protect memory owned by a VM, CloudVisor tracks memory pages of a VM and encrypts page content upon unauthorized mappings from the VMM and other VMs. The privacy of Disk I/O data is protected using whole virtual disk encryption: disk I/O between VMM and guest VMs are intercepted and encrypted on disk write and decrypted on disk read. To defend against tampering with encrypted pages and persistent storage data, CloudVisor uses the MD5 hash algorithm and Merkle tree [42] to do integrity checking of disk data during decryption.

In the software stack, only CloudVisor is within the trusted computing base, while other software such as the VMM and the management VM are untrusted. The integrity of CloudVisor can be ensured using the authenticated boot provided by the trusted platform module (TPM) [66]. To simplify the development and deployment of CloudVisor, we leverage Intel Trusted eXecution Technology (TXT) [27], which allows launching and measuring the CloudVisor after the platform has been initialized. In this way, CloudVisor is freed from most hardware initialization work.

We have designed and implemented a prototype system, based on commercially-available hardware support for virtualization, including ISA virtualization (i.e., VT-x [45]), MMU virtualization (i.e., EPT), I/O virtualization (e.g., IOMMU [5], SR-IOV [18]) and dynamic platform measurement (e.g., Intel Trusted eXecution Technology [27]). CloudVisor comprises around 5.5K LOCs and supports running mostly unmodified[1] Xen VMM with multiple Linux and Windows as guest OSes. Our performance evaluation using a range of benchmarks shows that CloudVisor incurs moderate slowdown for I/O intensive applications (ranging from 4.5% to 54.5%) and very small slowdown for other applications (ranging from 0.1% to 16.8%) compared to vanilla Xen.

[1]An *optional* patch with about 100 LOCs to reduce unnecessary *VM exits*, similar to the optimization in Turtles [13].

In summary, this paper makes the following contributions:

- The case of using *nested virtualization* to separate security protection from resource management of virtualization, which is backward-compatible with commercial virtualization stack and significantly reduces the TCB size from millions lines of code to only several thousand lines of code.
- A set of protection techniques that provide whole VM protection against adversaries who are even with full control of a VMM and the management VM.
- A prototype implementation that leverages existing hardware support for virtualization, which is demonstrated with low performance overhead.

The rest of this paper is organized as follows. The next section identifies threats to virtualized multi-tenant cloud and describes the threat model under CloudVisor. Section 3 first discusses our design goals, and then describes our approaches as well as the overall architecture of CloudVisor. Section 4, 5 and 6 describe how CloudVisor secures CPU states, memory pages and disk storages accordingly. The implementation issues and status are discussed in section 7. We then present performance evaluation results in section 8, and discuss the current limitation and possible future work in section 9. Finally, section 10 relates CloudVisor with other literatures and section 11 concludes this paper.

2. MOTIVATION AND THREAT MODEL

This section first identifies the attack surface of a virtualized multi-tenant cloud and then discusses the threat model under CloudVisor.

2.1 Attack Surface of Virtualization Layer

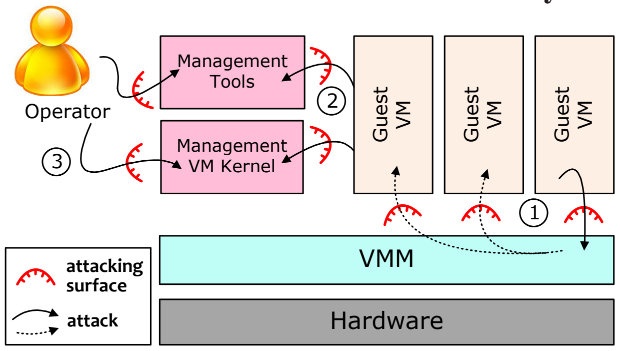

Figure 1: Typical virtualized architecture and attack surface in multiple tenant cloud.

Virtualization [24] has a good engagement with cloud computing due to its features in server consolidation, power saving and eased management. Many cloud providers have used virtualization in its cloud infrastructure and leasing resources to users in the form of virtual machine (a form of "Infrastructure as a Service" cloud), such as Amazon EC2 [6], Eucalyptus [46], FlexiScale [20] Nimbus [64] and RackSpace Cloud [62].

Virtualization might have both positive [16] and negative [22] impacts on the security and trustworthiness of the cloud. On the positive side, many "out-of-the-box" security techniques could now be implemented in the virtualization layers, making them more resilient to attacks to the VM [16]. On the negative side, commodity virtualization software stack is usually huge and most of them are within the trusted computing base.

Figure 1 depicts the typical (hostless) architecture of virtualization and the attack surface in a multi-tenant cloud. As tenant VMs are usually managed by the management tools via over-powerful privileged interfaces to the VMM, they could be arbitrarily inspected and tampered with by *not only the VMM but also the management tools in the management VM*. In principle, operators should be granted with only the least privilege and will not be able to tamper with tenant VMs. In practice, however, operators are usually granted with access rights more than they should have, as it is usually difficult to define the proper privilege precisely [35]. Consequently, improperly granting access rights to users' data could easily put users' data under threat (i.e., attack surface 3). For example, a cloud operator might leverage the internal maintenance interface to dump a VM's memory image for offline analysis, stealthily migrate/clone a VM to a shadow place for replaying, or even copy away all VM's virtual disks.

Worse even, as more and more functionalities being integrated into the virtualization layer such as live migration, security monitoring and snapshot, the TCB, which *includes VMM, management VM and management tools*, is exploding in both size and complexity. For example, the TCB size for Xen, including the VMM, management VM and tools has been steadily increasing across each major release, as shown in Table 1. An adversary could mount attacks to the virtualization layer by exploiting the inside security vulnerabilities (attack surface 1 and 2). Here, we deliberately separate internal (surface 3) and external attacks (surface 1 and 2) as in typical datacenter there are usually physically separated network for internal operators and for external accesses. Usually, internal attacks are much more powerful and easy to mount if a cloud operator tends to be malicious.

However, most previous efforts only aim at protecting against attack surface 1 and 2 by securing [60, 68] or removing [31] the virtualization layer, without defending attackers leveraging attack surface 3. For example, they cannot defend against attacks leveraging legal maintenance operations such as dump/clone/migrate a VM or virtual disks. Further, they require a reconstruction of the cloud software stack. To this end, it is critical to provide multi-tenant cloud with an approach that defending against attackers penetrated through the three attack surfaces from tampering with tenant VMs, yet with a small trusted computing base, which motivates the design and implementation of CloudVisor.

	VMM	*Dom0 Kernel*	*Tools*	TCB
Xen 2.0	45K	4,136K	26K	4,207K
Xen 3.0	121K	4,807K	143K	5,071K
Xen 4.0	270K	7,560K	647K	8,477K

Table 1: TCB of Xen virtualization layer (by Lines of Code, counted by *sloccount*).

2.2 Assumptions and Threat Models

Adversaries: Given that there are multiple attack surfaces in a multi-tenant cloud, we consider both local adversaries and remote adversaries and assume that they have full control over the VM management stack including the commodity hypervisor, the management VM and tools. An adversary may leverage the powerful management interfaces to try to dump a tenant VM's memory, steal the VM's virtual disks, or even inject code to the VM.

Assumptions: We assume the *cloud provider* itself does not intend to be malicious or with the goal of tampering with or stealing its tenant's sensitive information. Instead, the threat may come from the intentional or unintentional mis-operations from its operators [26, 63]. Hence, we assume there will be no internal physical attacks such as placing probes into the buses and freezing all main memory and reading out the data. Actually, typical data-centers usually have strict control of physical accesses as well as surveillance cameras to monitor and log such accesses. However, as the disk storage might be easily accessible by operators through the VM management stack or even physical maintenance (such as disk replacements), we assume that the external disk storage is not trustworthy.

Security Guarantees: The goal of CloudVisor is to prevent the malicious VM management stack from inspecting or modifying a tenant's VM states, thus providing both *secrecy* and *integrity* to a VM's states, including CPU states, memory pages and disk I/O data. CloudVisor guarantees that all accesses not from a VM itself (e.g., the VMM, other VMs), such as DMA, memory dumping and I/O data, can only see the encrypted version of that VM's data. Upon illegal tampering with a VM's states, CloudVisor uses cryptographic approaches to verify the integrity, ordering and freshness of a VM's data and fail-stops a VM upon tampering.

A malicious VMM cannot issue arbitrary control transfers from the VMM to a tenant' VM. Instead, all control transfers between the VMM and a VM can only be done through a well-defined entry and exit points, which will be mediated by CloudVisor. The VMM cannot fake an execution context to let a VM run upon. Actually, a VM's execution context is securely saved and restored by CloudVisor during a control transfer.

With platform measurement techniques such as Intel Trusted eXecution Technology and TPM, CloudVisor allows cloud tenants to assure that their VMs are running "as is" on machines protected by CloudVisor. Hence, attackers cannot alter the booting environment or fool a tenant's VM to run in a wrong execution mode such as a para-virtualized mode and a different paging mode, which will be detected and refused by CloudVisor.

Non-Security Goals: As a tenant's VM still uses services provided by the VMM and its management VM and tools, CloudVisor cannot guarantee availability and execution correctness of a tenant's VM. However, we believe this is not an issue for multi-tenant cloud, as the primary goal of cloud providers is featuring utility-style computing resources to users with certain service-level agreement. Providing degraded or even wrong services will be easily discovered by customers and the misbehaving provider or operator will soon be dumped out of the market.

CloudVisor does not guard against side-channel attacks in the cloud [49], which may be hard to deploy and have very limited bandwidth to leak information. However, CloudVisor does leverage advanced hardware features like AES instructions in recent CPUs [4] to prevent leakage of crypto keys [56]. Further, many security-critical applications such as OpenSSL have builtin mechanism to defend against side-channel attacks.

CloudVisor also provides no protection to interactions of a VM with its outside environments. Hence, the security of a tenant's VM is ultimately limited by the VM itself. For example, an adversary may still be able to subvert a VM by exploiting security vulnerabilities inside the VM. This can be usually mitigated by leveraging the traditional security-enhancing mechanisms for applications and operating systems. CloudVisor does guarantee that, adversaries controlling a subverted VM or even having subverted the management software or the VMM, cannot further break the security protection by CloudVisor to other VMs in the same machine.

3. GOALS AND APPROACHES

This section first illustrates the design goals of CloudVisor, and then describes approaches to achieving the goals. Finally, we present the overall architecture of CloudVisor.

3.1 Design Consideration

The primary goal of CloudVisor is to provide transparent security protection to a whole VM under existing virtualized platforms, yet with minimal trusted computing base:

Whole-VM protection: We choose the protection granularity at the VM level for three considerations. First, many cloud platforms such as Amazon EC2 choose to provide tenants with resources in the form of VMs (i.e., Infrastructure as a Service). Second, the VM is with a simple and clean abstraction and the interactions between a VM and the VMM are well-defined, compared to those between a process and an operating system, which usually is with several hundreds to thousands of APIs with complex and subtle semantics (e.g., ioctl). Finally, protection at the VM level is transparent to guest OS above. By contrast, providing protection at the process level (e.g., CHAOS [15, 14], Overshadow [17] and SP^3 [71]) is usually closely coupled with a specific type of operating system and requires non-trivial efforts when being ported to other operating systems.

Non-intrusive with Commodity VMMs: It is important to design a security-enhancing approach to working non-intrusively with existing commercially-available virtualization stack. Hence, CloudVisor should require minimal changes to both the VMM and the management software. This could enable rapid integration and deployment of CloudVisor to existing cloud infrastructure. Further, CloudVisor can then be separately designed and verified, and be orthogonal to the evolvement of the VMM and management software.

Minimized TCB: Prior experiences show that a smaller code size usually indicates more trustworthy software [21, 58]. Hence, the TCB size for CloudVisor should be minimal so that CloudVisor could be verified for correctness. For example, recent formal verification effort [34] has shown its success in a general-purpose OS kernel with 8,700 LOCs.

3.2 Approach Overview

Unlike traditional virtualization systems, CloudVisor excludes a VMM and the management VM out of the TCB. Instead, CloudVisor executes in the most privileged mode of a machine and monitors the execution of and interactions between the VMM and the hosted VMs, both of which execute in less privileged modes. As the resources of a VM mainly comprise of CPU, memory and I/O devices, CloudVisor is designed to protect such resources accordingly (as shown in Table 2):

Transparent Interposition using Nested Virtualization: To make CloudVisor transparent with existing virtualization stack, we use nested virtualization [23, 13] to give the illusion that a VMM

Category	Protecting Approaches
CPU states	Interpose control transfers between VMM and VM Conceal CPU states from the VMM
Memory pages	Interpose address translation from guest physical address to host physical address
Persistent data	Transparent whole VM image encryption Decrypt/encrypt I/O data in CloudVisor
Bootstrap	Intel TXT to late launch CloudVisor Hash of CloudVisor is stored in TPM

Table 2: Methodologies to protect a tenant VM.

still controls all resources of VMs. To achieve this, CloudVisor interposes all control transfer events between the VMM and its VMs (section 4). Upon interposition, CloudVisor does necessary transformation and protection, and forwards the (virtualized) events to the VMM to handle. For example, upon an interrupt and depending on the context, CloudVisor will save general-purpose registers and only provide necessary ones to the VMM, to limit information being exposed to the VMM.

VM-based Memory Ownership Tracking: To protect a VM's memory from inspection by the VMM and the management VM, CloudVisor interposes address translation from guest physical address to host physical address. Specifically, CloudVisor tracks the ownership of each page and each page table maintained by the VMM (i.e., extended page table, EPT [2]) (section 5). CloudVisor disallows the VMM from directly overwriting the EPT. On intercepting updates to the VMM's page table, CloudVisor checks if the ownership of the page matches with that of the page table and encrypts the page content if there is a mismatch.

One alternative approach to protecting a guest VM's memory might be multi-shadowing [17], which provides both encrypted version (seen by the VMM) and plain version (seen by the guest VM) of a page. However, this would require two EPTs for each VM and two copies of some pages, which causes additional memory pressure. Further, the VMM sometimes needs to access some guest VMs' memory in plain form, which requires interposition and protection by CloudVisor (section 5.3). Simply providing encrypted versions of pages to the VMM would corrupt the whole system.

I/O Protection through Encryption: CloudVisor currently provides protection to virtual disks owned by a VM. For network devices, as typical security-sensitive applications have already used encrypted message channels such as SSL, CloudVisor does not provide cryptography protection to such devices. To protect virtual disks, CloudVisor transparently encrypts and decrypts data during each disk I/O access by a VM, including both port-based I/O and direct memory access (DMA) (detailed in section 6). The integrity of disk data is ensured using the MD5 hash algorithm and Merkle tree [42] to do integrity checking (section 6). To prevent a VM, the VMM or the management VM from issuing DMA attacks, CloudVisor maintains a per-VM I/O access permission table (i.e., by manipulating the IOMMU [3]) and only grants DMA accesses to their own memory regions.

Late Launch to Reduce CloudVisor Complexity: As CloudVi-

[2]Translates guest physical address to host physical address.

[3]IOMMU translates the guest physical I/O addresses to host physical addresses on an memory access issued by I/O devices.

sor runs underneath the VMM, CloudVisor has to implement many machine initialization procedures if it is booted before the VMM. This could increase the complexity and also the code base of CloudVisor. Hence, CloudVisor leverages existing hardware support for dynamic measurement [27, 9] and boots CloudVisor after the system has finished its booting process. Specifically, upon receiving requests of booting CloudVisor from the VMM, CloudVisor boots itself and the processor will issue a measurement on the integrity of CloudVisor, which prevents the VMM from booting a tampered version of CloudVisor. The measurement results will be used by cloud tenants as evidences in remote attestation.

3.3 The CloudVisor Architecture

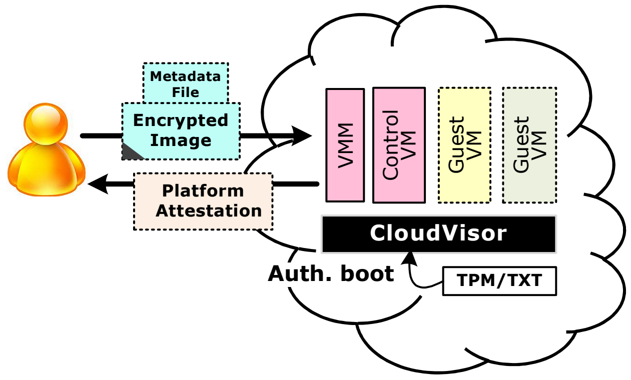

Figure 2: Overall architecture of CloudVisor.

Figure 2 shows the overall architecture of CloudVisor, which is a featherweight security monitor that runs at the most privileged level, while the commodity VMM is deprivileged into the less privileged mode together with the control VM and guest VMs. CloudVisor enforces the isolation and protection of resources used by each guest VM and ensures the isolation among the VMM and its guest VMs. Traditional virtualization functionalities, such as resource management, VM construction and destruction, scheduling, are still done by the VMM. CloudVisor transparently monitors how the VMM and the VMs use hardware resources to enforce the protection and isolation of resources used by each guest VM.

Figure 2 also depicts how cloud users could use CloudVisor to securely deploy their services. A cloud tenant may first authenticate the cloud platform by using TCG's attestation protocol with TPM to know if the platform is running a known version of CloudVisor. Then, the tenant may send VM images and the corresponding metadata file to run in the cloud. Similar to Amazon Machine Images [7], the image is encrypted using a random symmetric key. The public key will then be used to encrypt the symmetric key and the users will send both cipher-texts to CloudVisor. CloudVisor controls the private key of the platform and uses it to decrypt the images for booting. In the metadata file for the VM image, there is some information (such as hashes and initial vectors) guarding the integrity, ordering and freshness of the VM images. The metadata also contains information describing the execution modes (e.g., paging mode) of this VM. Upon launching of a VM, CloudVisor will use this information to ensure that the VM image is executed "as is".

4. SECURING CONTROL TRANSITION WITH NESTED VIRTUALIZATION

CloudVisor interposes control transitions between a VMM and its guest VMs. With hardware support for virtualization (i.e., VT-x [45] or SVM [9]), such control transitions are abstracted with *VM exit* (transitions from a VM to the VMM) and *VM entry* (transitions from the VMM back to a VM). CloudVisor transparently secures such transitions using nested virtualization [23, 13] by virtualizing such events and doing necessary security protection. This section first introduces the necessary background information with hardware-assisted (nested) virtualization using Intel's VT-x as an example, and then describes how CloudVisor leverages it to secure control transitions.

4.1 Hardware-assisted (Nested) Virtualization

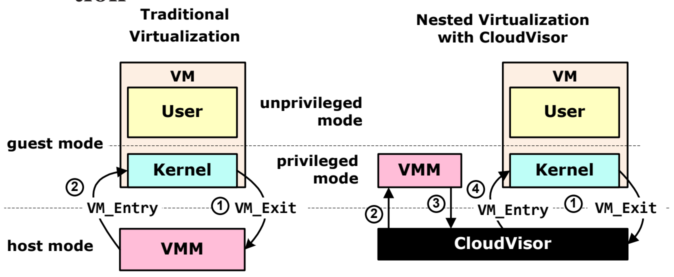

Figure 3: The general architecture of hardware-assisted virtualization (left) and how CloudVisor leverages it to secure control transfer using nested virtualization.

The left part of Figure 3 shows the general architecture of hardware-assisted virtualization, where the VMM runs in *host mode*, and the VMs run in *guest mode*. The former mode is used by the VMM and instructions are natively executed. The latter mode is used by guest VMs, where privileged instructions that access critical hardware resources (e.g., I/O resources) will cause a control transition from *guest mode* to *host mode* (a *VM exit*, step 1). The VMM will handle the event (e.g., by emulating the violating instruction) and then use *VM entry* to transfer the control back to *guest mode* (step 2), where the guest VM resumes its execution.

For each virtual CPU of a guest VM, an in-memory VM control structures (*VMCS* in Intel's terminology) is maintained by the VMM. The *VMCS* saves the states for the VMM and the guest VM, as well as controls which guest events should cause *VM exit*.

With nested virtualization, CloudVisor now runs in *host mode*, while both the VMM and the guest VMs are put in *guest mode*, as shown in the right part of Figure 3. To enforce isolation between a VMM and its guest VMs, the VMM runs in a separated context of *guest mode*. Note that, placing the VMM into a less privileged mode will not degrade the security of the VMM, as CloudVisor will ensure strict isolation among the VMM and its VMs.

4.2 Securing Control Transition with Nested Virtualization

Enabling Interposition: CloudVisor maintains a *VMCS* for the VMM to control the types of instructions or events that will cause *VM exit* when executing in the VMM's context. Currently, the VMM only gets trapped on three types of architectural events relating to resource isolation: 1) NPT/EPT faults, which are caused by faults on translation from guest physical address to host physical address; 2) Execution of instructions in the virtualization instruction set such as VMRead/VMWrite [4]; 3) IOMMU faults, which is caused by faults during the translation from device address to host physical address. Other architectural events like page faults and interrupts do not cause traps to CloudVisor and are directly delivered to the VMM.

The VMCS for each guest VM is still created and maintained by the VMM. When a VMCS is to be installed on the CPU, CloudVisor overwrites some critical control fields. For instance, the entry address of *VM exit* handler is specified in the *VMCS*. To interpose control transition, CloudVisor records the entry address and replaces it with the entry address of the handler in CloudVisor. As a result, all *VM exits* from a guest VM is first handled by CloudVisor and then propagated to the VMM.

Securing Control Transition: CloudVisor interposes between guest VMs and the VMM on *VM exit* for mainly three purposes: 1) protecting CPU register contexts when a VM is interrupted; 2) manipulating address translation to enforce memory isolation (detailed in section 5); 3) intercepting and parsing I/O instructions to determine the I/O buffer addresses in a VM (detailed in section 6).

As shown in the right part of Figure 3, CloudVisor interposes each *VM exit* event (step 1), protects CPU contexts and parses I/O instructions if necessary, and then forwards the *VM exit* event to the VMM (step 2). It then intercepts the *VM entry* request from the VMM (step 3), restores CPU contexts and resumes the execution of guest VM (step 4) accordingly.

Both external interrupts and certain instruction execution can cause *VM exits*. For external interrupts, the VMM does not need the general-purpose registers to handle the event. In that case, CloudVisor saves and clears the content of general-purpose registers before propagating the event to the VMM. On *VM entry*, CloudVisor restores the saved registers for the guest VM and resumes the VM.

For *VM exits* caused by synchronous instruction execution, CloudVisor only resets a part of the register contexts and keeps the states that are essential for the event handling. For instance, the program counter and some general-purpose registers in an I/O instruction should be exposed to the VMM.

CloudVisor ensures the CPU context on *VM entry* is exactly the same with the context on last *VM exit* for each virtual CPU. Hence, the VMM is unable to dump CPU register information by triggering arbitrary interrupts, redirect control to arbitrary code in the guest VM, or tamper with the CPU context of the guest VMs.

4.3 Dynamic Nested Virtualization

Though, CloudVisor runs underneath the VMM, CloudVisor does not contain machine bootstrap code for the sake of small TCB. Consequently, it is booted after the VMM and the management VM have been initialized. When CloudVisor boots, it runs in *host mode* and demotes the VMM to *guest mode*, thus effectively virtualizes the VMM on the fly. To ensure a tamper-proof dynamic nested virtualization, CloudVisor adopts dynamic root of trust (such as Intel TXT [27] and AMD SVM [9]) to ensure the processors are in a known clean state when they initialize CloudVisor. The SHA-1 hash of CloudVisor binary is calculated and stored in the TPM [66] for future remote attestation. This is done in the macro instruction such as SINIT (Intel TXT) and SKINIT (AMD SVM) that are hardwired in the processor. For multi-processor or multi-core platforms, all the processors are synchronized before launching CloudVisor to ensure the all the processors are nestedly virtualized simultaneously.

[4]`VMRead` and `VMWrite` instructions read/write VM control structures (*VMCS*)

5. MEMORY ISOLATION

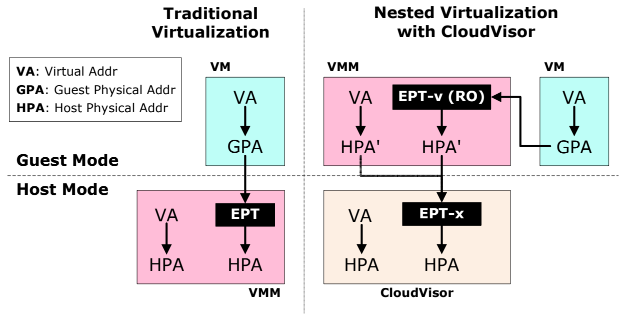

Figure 4: The general structure of extended paging (left) and how CloudVisor leverages it for memory isolation.

To provide efficient memory isolation among CloudVisor, the VMM and guest VMs, CloudVisor uses commercially-available extended paging or nested paging, which provides hardware support for MMU virtualization.

5.1 Isolation with Nested/Extended Paging

The left part of Figure 4 shows the intended usage of extended paging in virtualized systems: the VMM itself uses a translation table that directly converts virtual addresses (VA) to host physical addresses (HPA) and controls how VMs translate guest physical addresses (GPA) to HPA using an extended page table (EPT). The guest VM manages the address translation from VA to GPA with the conventional page table.

When CloudVisor is booted, the VMM is demoted to run in the *guest mode*. An extended page table (EPT) is created for the VMM and the address translation of the VMM is then configured to use a two-step address translation that uses page table (VA to GPA) and extended page table (EPT) (GPA to HPA). As shown in the right part of Figure 4, CloudVisor maintains an identity GPA-to-HPA mapping (i.e., HPA' equals to HPA) EPT for the VMM (called *EPT-x*). Thus, the VMM is unaware of the memory virtualization by CloudVisor. CloudVisor removes its own memory from EPT-x to isolate its memory space from the VMM. *EPT-x* is kept in the memory space of CloudVisor and is not accessible by the VMM. The VMM still maintains a GPA-to-HPA' mapping table (called *EPT-v*) for each VM, but is granted with only read permission.

In principle, a guest VM can be configured to use either software address translation such as shadow page table or hardware-assisted address translation. The support of software address translation should be technically doable in CloudVisor but might be more complex. For simplicity, CloudVisor currently only supports platforms with hardware-assisted address translation. If the VMM tricks a guest VM into using a software address translation mechanism, CloudVisor will refuse to attest for the VM.

5.2 Memory Ownership Tracking

To ensure memory isolation among the VMM and its guest VMs, CloudVisor maintains a table to track the ownership of each physical memory page. The value of the table is the owner ID of the page. Each VM is assigned with a unique ID when it is booted. The VMM's ID is fixed to zero. CloudVisor ensures that a physical memory page can only be assigned to be one owner at a time.

During system startup, all pages other than those in CloudVisor are owned by the VMM. When the EPT of a guest VM is loaded into the processor for the first time, CloudVisor walks through the whole EPT to find all the mapped pages. These pages are regarded as being assigned to the guest VM. CloudVisor changes the owner of these pages to the guest VM, and unmaps it from the EPT of the VMM so that the VMM cannot access the pages any more. When a page is unmapped from the EPT, the owner of the page is set to be the VMM and the page is mapped back in the EPT of the VMM.

Whenever the VMM updates the guest EPT, a page fault in the EPT (EPT violation in Intel's term) is raised. CloudVisor handles the fault by validating the page ownership. If a new mapping is to be established, CloudVisor ensures that the page to be mapped belongs to the VMM. CloudVisor unmaps it from the EPT of the VMM and changes the page owner to the guest VM. If an existing page is to be unmapped, CloudVisor encrypts the content of the page, maps it to the EPT of the VMM and changes the page owner to the VMM. CloudVisor does not allow a page to be mapped in the same EPT more than once. To remap a page, the VMM has to unmap it first, and then remap it to the new location.

DMA-capable devices can bypass memory access control enforced by MMU. To defend against malicious DMA requests, CloudVisor makes protected memory regions inaccessible from DMA devices using IOMMU by manipulating the translation from host physical address to device address. During system startup, CloudVisor unmaps its own memory in the IOMMU table to prevent DMA requests from accessing the memory. When a guest VM boots up, CloudVisor also unmaps the VM's memory in the IOMMU table used by DMA-capable devices. When the guest VM shuts down, the pages are returned to the VMM and CloudVisor remaps the pages in the IOMMU table. If a DMA request is setup to access memory pages in CloudVisor or guest VMs, an IOMMU fault is raised and handled by CloudVisor. Currently, CloudVisor simply denies the request.

5.3 Legal Memory Accesses

Memory isolation mechanism provided by CloudVisor ensures the entire memory space of a guest VM is inaccessible to the VMM and the management VM. However, there are several cases where the VMM and the management VM should be allowed to access some memory of guest VMs. In such cases, CloudVisor interposes and assists such accesses to ensure that only minimal insensitive information will be divulged.

Privileged instructions such as I/O instructions and accesses to control registers cause traps (i.e., *VM exits*) that are handled by the VMM. In some cases the VMM needs to get the instruction opcode in the guest VM memory to emulate it. During such traps, CloudVisor fetches the privileged opcode and feeds it to the VMM. As CloudVisor only allows fetching one opcode pointed by the program counter, the VMM is unable to trick CloudVisor into fetching arbitrary non-privileged opcode, nor can it arbitrarily trigger traps to access opcode.

On a trap, the program counter of the faulting instruction is a virtual address and the memory operands are also presented as virtual addresses. The VMM needs to walk the page table in the guest VM to translate the virtual addresses to guest physical addresses, which are further translated to host physical addresses using the EPT. To handle this, CloudVisor temporarily allows the VMM to indirectly read the guest page table entries corresponding to the opcode and memory operands. Upon a trap caused by the execution of a privileged instruction, CloudVisor fetches the program counter of the instruction and parses the instruction to get the memory operand. CloudVisor walks the page table in the guest VM to get the page table entries required to translate the program counter and the memory operands. When the VMM accesses the page table, CloudVisor feeds it with the previously obtained page table entries. To reduce overhead associated with privileged instruction emulation, CloudVisor uses a buffer to cache the page table entries for privileged instructions for each VCPU.

The VMM also needs to get the contents of guest I/O buffers when emulating I/O accesses. When the VMM accesses I/O buffers, an EPT fault is raised and CloudVisor handles the fault by copying the data for the VMM. Specifically, when the VMM copies data to or from the guest VM, CloudVisor validates that the buffer address in the guest VM is a known I/O buffer and determines if the buffer is used for disk I/O (section 6.1).

6. DISK STORAGE PROTECTION

Virtual disks of a VM are also critical resources that demand both privacy and integrity protection. There are two alternative ways to protect a virtual disk. The first one is letting a cloud user use an encrypted file system that guard the disk I/O data at the file-system level, such as Bitlocker [1] and FileVault [3]. This requires no protection of disk I/O data by CloudVisor as the disk I/O only contains encrypted data and the file system itself will verify the integrity and freshness of disk data.

To provide transparent protection to tenant's VM, CloudVisor also provides full-disk encryption and hashing to protect disk data privacy and integrity. CloudVisor encrypts the data exchange between a VM and the VMM and verifies the integrity, freshness and ordering of disk I/O data.

6.1 Handling Data Exchange

Retrieving I/O configuration information: When a VM boots up, the guest VM usually probes its I/O configuration space (e.g., PCI) to identify I/O devices and their ports. The VMM usually virtualizes the I/O space and feeds the VM with information of the virtual devices. CloudVisor interposes the communication to gather the information of virtual devices plugged into the guest VM. In this way, CloudVisor knows the I/O ports used by the VM and their types. Among the I/O ports, CloudVisor treats disk I/O ports differently from others such as VGA, network and serial console. All data exchanged through disk I/O ports are encrypted and hashed before being copied to the VMM and decrypted before copying data to a guest VM. CloudVisor does not encrypt or decrypt data exchanges on other ports (such as NICs).

Interposing I/O requests: To determine if I/O data exchange between a guest VM and the VMM is legal, CloudVisor intercepts and parses I/O requests from the guest VM. CloudVisor does not emulate the I/O requests but only records the requests. By parsing I/O requests from the I/O instructions, CloudVisor retrieves the information of *I/O port*, *memory address* of the I/O buffer and the *buffer size* for further processing.

There are two alternative ways to process these I/O requests. The first one is to "trap and emulate" the requests. To ensure security, CloudVisor uses a *white-list* to record the I/O requests and then propagates them to the VMM, which will handle the requests by copying the data to/from buffers in the guest VM. The following data copying will trap (i.e., VM exit) to CloudVisor, which will use the *white-list* to validate the data copying. After the data exchange, the corresponding record will be removed from the list to prevent the VMM from revisiting the memory pages.

The second approach is using a *bounce buffer* in CloudVisor to assist the data exchange. When a guest VM tries to copy data to the VMM (i.e., an I/O write operation), CloudVisor intercepts the request, copies the data to the bounce buffer, and then provides the VMM with a modified I/O request to let the VMM read from the bounce buffer instead of the guest VM. Similarly, when a VMM tries to copy data to a guest VM, the data is first written to the bounce buffer and then read by the guest VM.

In principle, the first "trap and emulate" based approach can handle both port-based I/O and direct memory access (DMA). However, it will introduce a large amount of traps (i.e., VM exits) to CloudVisor if the data is copied in small chunks (e.g., 4 or 8 bytes). A DMA request that incurs multiple traps will cause non-trivial performance degradation for I/O intensive applications. Comparing to the first approach, the *bounce buffer* approach only incurs one additional data copy. Hence, CloudVisor uses the first approach for port-based I/O and the second approach for DMA.

6.2 Disk I/O Privacy and Integrity

CloudVisor uses the AES-CBC algorithm to encrypt and decrypt disk data in the granularity of disk sectors. A 128-bit AES key is generated by the user and passed to CloudVisor together with the encrypted VM image. The storage AES key is always maintained inside CloudVisor.

At VM bootup time, CloudVisor fetches all non-leaf nodes of hashes and IVs (Initial Vectors) in the Merkle hash tree and keeps them as in-memory cache. On disk reads, CloudVisor first hashes the data block to verify its integrity and then decrypts the disk sector using the AES storage key and the IV. On disk writes, an IV is generated for each disk sector if it has not been generated yet. The data block is hashed and then encrypted using the storage key and the IV.

As CloudVisor does not control the devices, it cannot read or write metadata on external storage space. One approach is let CloudVisor issue shadow DMA requests to the VMM. However, our experience shows that this sometimes could incur timeout for I/O requests in a guest VM. Instead, we provide a user-level agent in the management VM to assist metadata fetching, updating and caching. The agent is untrusted as it has the same privilege as other software in the management VM. If the agent refuses to function or functions incorrectly, CloudVisor can always detect the misbehavior by validating the metadata.

As shown in Figure 5, for each disk sector, a 128-bit MD5 hash and a 128-bit IV are stored in a file stored in the file system of the management VM. The hash is organized using a Merkle tree to guard the freshness and ordering of disk data. When launching a guest VM, the file is mapped into the memory of the agent. The agent fetches all the non-leaf hashes in the Merkle hash tree and sends them to CloudVisor. CloudVisor caches the hashes in memory to eliminate further fetches.

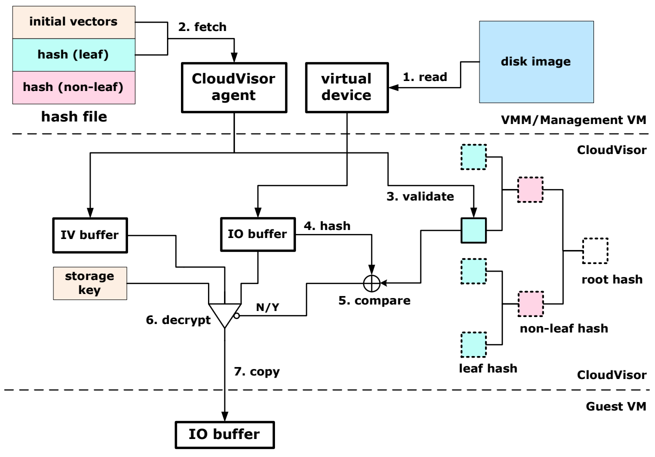

Figure 5: Data flow of disk I/O read.

On a disk I/O read, the virtual device driver first reads the requested disk block from the disk image of the VM (step 1). Then the agent fetches the hash and IV of the requested disk block and puts them in the hash and IV buffer provided by CloudVisor (step 2), and the integrity of the fetched hash is validated by CloudVisor (step 3). The MD5 hash of the cipher text is calculated (step 4) and compared with the fetched hash (step 5). If the computed hash matches with the store hash, CloudVisor decrypts the data using the fetched IV and the storage key (step 6). If the data is valid, CloudVisor copies it to the I/O buffer in the guest VM and removes the buffer address from the *white-list* if necessary (step 7). The disk I/O write operation is similar to read. The difference is that CloudVisor will generate the IVs and hashes and put it to metadata file.

The agent leverages the file cache of the operating system to buffer the most frequently used metadata. In the worst case, for a disk read, the agent needs two more disk access to fetch the corresponding hash and IV.

Sudden power loss may cause state inconsistency, as CloudVisor currently does not guarantee atomic updates of disk data, hashes and IVs. For simplicity, CloudVisor assumes the cloud servers are equipped with power supply backups and can shutdown the machine without data loss on power loss. Recent researchers [70] have also shown that providing atomicity and consistency in a secure file system is not very difficult. We plan to incorporate such support in future.

7. IMPLEMENTATION ISSUES AND STATUS

CloudVisor has been implemented based on commercially-available hardware support for virtualization, including VT-x [45], EPT [45], VT-d [5], and TXT [27]. To defend against cache-based side-channel attacks among VMs [56], CloudVisor uses the AES-NI instruction set in CPU [4] to do encryption. For simplicity, CloudVisor currently only supports hardware-assisted virtualization. CloudVisor supports Xen with both Linux and Windows as the guest operating systems and can run multiple uniprocessor and multiprocessor VMs.

7.1 Multiple VMs and Multicore Support

CloudVisor supports multiple VMs to run simultaneously atop a multiprocessor machine. To support multiprocessor, CloudVisor maintains one VMCS for each CPU core used by the VMM. All CPU cores shares one EPT for the VMM (i.e., EPT-x) and CloudVisor serializes accesses from multiple cores to EPT-x. During startup, SINIT/SKINIT uses IPIs (Inter-processor Interrupt) to broadcast to all CPU cores and launch CloudVisor on all the cores simultaneously.

7.2 VM Life-cycle Management

CloudVisor can transparently support VM construction and destruction, as well as VM save, restore and migration. The followings briefly describe the involved actions required by CloudVisor to provide protection to a VM on these operations:

VM construction and destruction: Even if the whole VM image is encrypted, VM construction can still be supported by CloudVisor in a transparent way. I/O data are transparently decrypted when it is copied to guest VM memory. On VM destruction, the access right of memory pages of the VM is restored to the VMM transparently.

VM snapshot, save, restore: In order to support VM snapshot, save and restore, guest VM memory integrity and privacy should be guaranteed when it is on storage space. CloudVisor uses per-page encryption and hashing to protect memory snapshot. Similar to the protection on the disk image, the memory contents are hashed at the granularity of page. The hashes are organized as a Merkle tree and stored together with an array of IVs.

When the management VM maps and copies the VM memory for saving, the operation is not initiated by I/O instructions of the VM itself. CloudVisor would not be able to find a corresponding entry in the *white-list*. In that case, CloudVisor would encrypt the requested page with the AES storage key and a newly generated IV. The VMM will get an encrypted memory image. When restoring the memory image, the VMM copies the encrypted memory image into guest memory space. CloudVisor finds the corresponding storage key of the VM, fetches the IVs and hashes of the memory image and decrypts the pages.

VM migration: VM migration procedure is similar to VM save and restore, but requires an additional key migration protocol. CloudVisor on the migration source and destination platform needs to verify each other before migrating the storage key and root hash. The verification procedure is similar to the remote attestation procedure between CloudVisor and the cloud user, which is a standard remote attestation protocol from TCG [52].

7.3 Performance Optimization

Boosting I/O with hardware support: CloudVisor currently supports virtualization-enabled devices such as SR-IOV NICs, direct assignment of devices to a VM and virtualizing traditional devices. For the first two cases, as most of operations are done in a VM itself without interventions from the VMM, CloudVisor mostly only needs to handle the initialization work and does very little work when the devices are functioning.

Reducing unnecessary VM exits: On Intel platform, we found that a large amount of *VM exit* events are due to *VM read* and *VM write* instructions. The VMM intensively uses these instructions to check and update the states of guest VMs. These instructions always cause *VM exits* if not being executed in *host mode*. The large amount of *VM exits* would bring notable performance overhead. To remedy this, CloudVisor provides an optional patch to Xen that replaces the *VM read* and *VM write* instructions with memory accesses to the *VMCS*, similar to the method used in [13]. Note that, adding an in-memory cache for VMCS will not introduce security vulnerabilities, as CloudVisor always validates the integrity of a *VMCS* when the VMCS is loaded into CPU.

7.4 Key Management

Currently, CloudVisor uses a simple key management scheme by using the cryptography key within VM images. The VM key is encrypted using the public key of a machine (e.g., storage root key) so that the VM images can only be decrypted and verified by a know version of CloudVisor verified using authenticated boot by the trusted platform module (TPM) [66] and Intel's TXT [27]. When a VM is being launched, the encrypted VM key will be loaded into CloudVisor's memory and decrypted using the public key of the platform (e.g., storage root key in TPM). The decrypted VM key will then be used to encrypt/decrypt the data exchange between the VMM and the guest VM. To ease users' deployment, we also provide a user-level tool to convert a normal VM image to the encrypted form and generate the metadata file. Note that the key management scheme is orthogonal to the protection means in CloudVisor and can be easily replaced with a more complex scheme.

7.5 Soft and Hard Reset

An attacker might issue a soft reset that does not reset memory and try to read the memory content owned by a leased VM. In such cases, the reset will send a corresponding *INIT* signal to the processor, which will cause a *VMX exit* to CloudVisor. CloudVisor will then scrub all memory owned by the leased VMs and self-destruct by scrubbing its own memory. Similarly, if the VMM or a VM crashes, CloudVisor will do the same scrubbing work upon intercepting the events. For a hard reset, as the memory will be lost due to power loss, CloudVisor simply post the hard rest signals to CPUs. In short, CloudVisor uses a fail-stop manner to defend against possible attacks during hard and soft reset as well as crashes.

7.6 Implementation Complexity

We have built a prototype of CloudVisor based on commercially-available hardware support for virtualization. Our prototype consists of a set of event handlers for privileged operations (e.g., *VM exit* and *EPT fault*), a tiny instruction interpreter, and an AES library for encryption. The code base is only around 5.5K lines of code (LOCs), which should be small and simple enough to verify. There is also an untrusted user-level CloudVisor agent in the QEMU module of Xen. The agent consists of around 200 LOCs and handles the management of hashes for pages and I/O data.

CloudVisor supports Xen with both Linux and Windows as the guest operating systems. CloudVisor is mostly transparent to Xen with an *optional* patch with about 100 LOCs to Xen for Intel platform. Multiple unmodified Linux and Windows VMs can run with uniprocessor or multiprocessor mode simultaneously atop CloudVisor. CloudVisor could also transparently support VMMs other than Xen, as it makes little assumption on the VMM, which will be our future work.

8. PERFORMANCE EVALUATION

We evaluated the performance overhead of CloudVisor by comparing it with vanilla Xen using a set of benchmarks. As we did not have a machine with all features required by the current implementation, we used two machines to demonstrate the functionalities of CloudVisor separately. The first machine is equipped with an Intel Core2 Quad processor. The chipset supports Intel TXT [27] and has a TPM chip installed, but without EPT support [45]. This machine is used to demonstrate how CloudVisor dynamically establishes a trusted execution environment after the boot of Xen. We use a Dell R510 server as the second machine, which is equipped with two SR-IOV NICs connected with one Gigabyte Ethernet. This machine has a 2.6 GHz 4-core/8-thread Intel processor with VT-x, EPT and AES-NI support and 8 GB physical memory. Although verified separately, the evaluation of functionality and performance should still be valid as TXT and TPM are only effective at booting time and VM launch time. CloudVisor has no interaction with TXT and TPM during normal execution time.

We compare the performance of Linux and Windows VMs runs upon CloudVisor, vanilla Xen-4.0.0. XenLinux-2.6.31.13 is used as Domain0 kernel. Each VM is configured with one or more virtual CPUs, 1 GB memory, a 4 GB virtual disk and a virtual NIC. The VMs run unmodified Debian-Linux with kernel version 2.6.31 and Windows XP with SP2, both are of x86-64 version.

The application benchmarks for Linux VMs include: 1) Kernel Build (*KBuild*) that builds a compact Linux kernel 2.6.31 to measure the slowdown for CPU-intensive workloads; 2) *Apache* benchmark (ab) on Apache web server 2.2.15 [10] for network I/O intensive workloads; 3) *memcached* 1.4.5 [19] for memory and network I/O intensive workloads; 4) *dbench* 3.0.4 [65] for the slowdown of disk I/O workloads. *SPECjbb* [59] is used to evaluate the server side performance of Java runtime environment in the Windows VM. To understand the overhead in encryption and hashing and the effect of the *VM read* and *VM write* optimization, we also present a detailed performance analysis using *KBuild* and *dbench*. We further evaluate the performance and scalability of CloudVisor by running *KBuild* in multicore and multi-VM configurations. Finally, we use *lmbench*-3.0 [41] to quantify the performance loss incurred in primitive operating system operations. Each test were ran five times and the average result using performance slowdown (calculated using (Time(new) - Time(old))/Time(old)) is reported. Throughout the evaluation, hardware cryptographic instructions (Intel AES-NI) are used to perform the AES encryption and decryption.

8.1 Performance of Uniprocessor VMs

We use two applications to quantify the performance slowdown of uniprocessor VMs underlying CloudVisor. For *KBuild*, we build a compact kernel by running “make allnoconfig” and record the time spent to complete the compilation. For *Apache*, we use Apache Benchmark (ab) to issue 10,000 requests with concurrency level 500 to request a 4 Kbyte file from a client machine, and collect its transfer rate. For *memcached*, we use a remote client to issue requests to a *memcached* server, which is listening to a UDP port. For *SPECjbb*, we use the standard testing script of *SPECjbb* to evaluate the average number of business operations for eight data warehouses on a Windows XP VM.

As shown in Figure 6, for *KBuild*, the performance slowdown is relatively high (6.0%), as there are some disk I/O requests associated when reading files from disks and writing files back. Disk I/O operations in CloudVisor requires interposition and cryptographic operations, which are the major source of slowdown. For *Apache*, as it mostly involves networked I/O which requires no intervention by CloudVisor, CloudVisor incurs only 0.2% slowdown. For *memcached*, the performance slowdown is also quite small (0.1%). This is because most of the operations are in memory and there are only a small amount of disk I/O requests. Hence, it rarely traps to CloudVisor. The performance overhead for *SPECjbb* is around 2.6% as it involes very few I/O operations. For all applications, most of the performance slowdown comes from the additional *VM exits* to CloudVisor and the emulation of some privileged instructions (such as disk I/O operations).

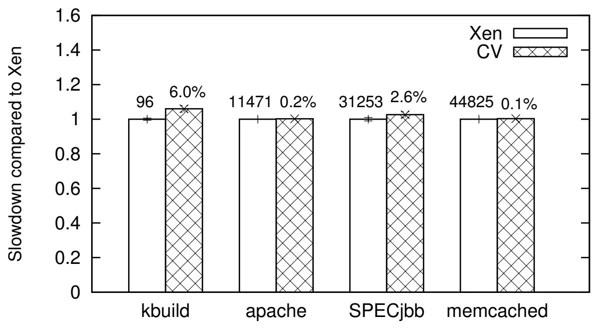

Figure 6: Slowdown for Kernel Build (*KBuild*), *Apache*, *SPECjbb* and *memcached*. The values shown in the bar indicate the execution time (s), transfer rate (KB/s), score and requests, accordingly.

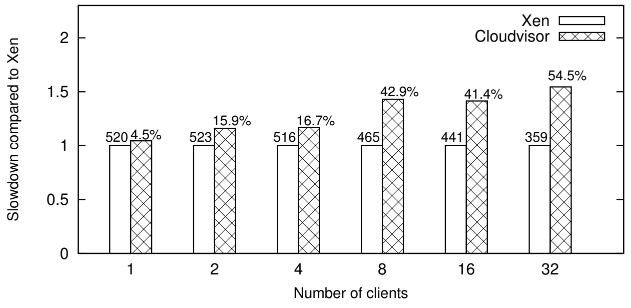

Figure 7: Performance slowdown of CloudVisor on *dbench* from 1 concurrent client to 32. The data on the left bar shows the raw throughput in MB/s.

I/O Performance: As disk I/O is likely a performance bottleneck in CloudVisor due to the cryptographic operations, we use *dbench* as a worst case benchmark to quantify the incurred slowdown. *dbench* mimics the I/O pattern of real applications that issues POSIX calls to the local filesystem, which taxes the I/O module in CloudVisor.

As shown in Figure 7, when the number of clients is small, *dbench* experiences relative small performance slowdown (4.5%, 15.9% and 16.7% for 1, 2 and 4 clients accordingly), due to the fact that the filesystem cache for IVs and hashes in the control VM has relatively better locality. However, when the number of clients is larger than four, the interference among clients causes worse locality for IVs and hashes, thus incurs large performance slowdown. This can either be fixed by increasing the filesystem cache in the control VM for a machine with abundant memory or integrating an intelligent cache in the untrusted user-level agent and CloudVisor.

Analyzing the Performance Slowdown: To understand the benefit of using the *VM exit* optimization, we profiled the execution of *KBuild*. We first collected the statistics of *VM exit* in CloudVisor and observed 4,717,658 *VM exits*, which accounted for around 3.31s out of 102s spent on *VM exit* handlers in CloudVisor. In contrast, before the *VM exit* optimization, we observed 9,049,852 *VM exits*. Hence, the optimization eliminated around 47.9% *VM exits*, which accounted for 2.11s in *VM exit* handlers.

As I/O intensive benchmarks stress the cryptographic module in CloudVisor, we used *dbench* running 32 concurrent clients to profile the slowdown caused by AES encryption/decryption and hashing. We replaced the AES cryptographic and/or hashing operations with NULL operations. Without encryption and hashing, the throughput of *dbench* is 270 MB/s. When only AES encryption for I/O data was turned on, the throughput is 255 MB/s, causing a relative slowdown of 5.9%. When both AES cryptographic and hashing were enabled, the throughput is 233 MB/s, causing a slowdown of around 9.4%. The evaluation showed that, with the new AES-NI instruction support, cryptographic operations incur only a small amount of slowdown, even when CloudVisor is highly stressed. Thus, the major overhead comes from the I/O interposition and metadata management, which are further worsened by poor locality due to interference among multiple clients.

8.2 Performance of Multiple VMs and Multi-core

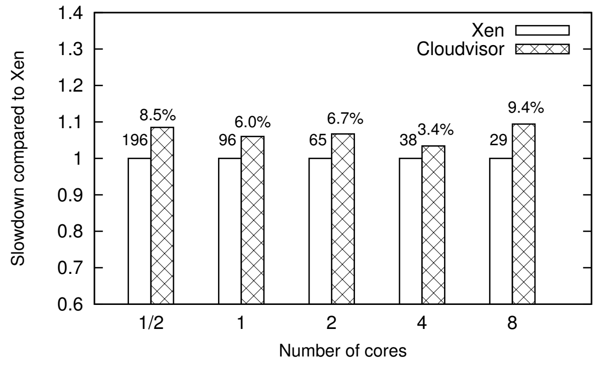

Figure 8: Slowdown of kernel build on a VM configured with 1/2, 1, 2, 4, 8 cores. When configured with 1/2 core, it means that there are two VMs runs on one core.

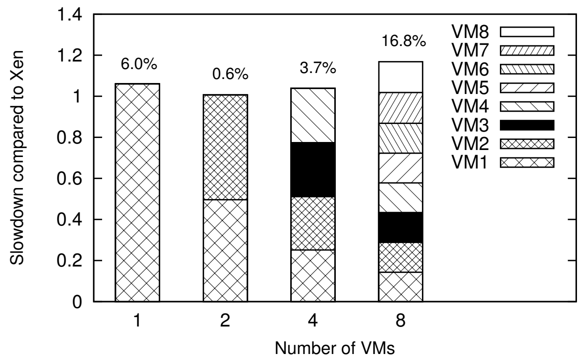

Figure 9: Slowdown of kernel build CloudVisor compared to Xen. The workload is concurrently run on multiple VMs.

To evaluate the performance of CloudVisor in multi-tenant cloud, we scaled up the system by increasing the number of cores managed by a VM or increasing the number of VMs. Figure 8 shows the scalability of a single VM running kbuild workloads on both Xen and CloudVisor. CloudVisor incurs at most 9.4% slowdown compared to Xen and the slowdown does not grow up much with the increase of processor cores. During the execution of a guest VM, *VM exits* are mainly caused by disk I/O requests.

The evaluation results of concurrent execution of *KBuild* on multiple VMs is shown in Figure 9. In the evaluation, each VM was configured with one core, 512 MB memory and one virtual disk. CloudVisor incurs at most 16.8% slowdown (8 VMs) compared to Xen. When the number of VMs is less than 8, the slowdown is moderate. In CloudVisor, disk I/O states of each guest VM are organized in the per-VM area that can be simultaneously accessed without lock protection. Multiple I/O requests can be handled by CloudVisor in parallel. However, with the increase of the number of VMs, the buffer cache used by the CloudVisor agent will be stressed. Although each guest VM has its own CloudVisor agent instance, the instances in the control VM share the file cache in the OS kernel.

8.3 OS Primitives:

app	*Xen*	*CV*	slowdown
ctx(16p/64k)	2.3	2.46	7.1%
stat	1.115	1.12	0.4%
mmap	2259	2287	1.2%
sh proc	1171	1437	22.7%
10k file(delete)	9.7	11.0	13.1%
Bcopy(hand)	3527	3443	2.4%
Bcopy(libc)	3565	3466	2.9%

Table 3: *The slowdown of OS primitives in CloudVisor (CV stands for CloudVisor).*

Lmbench is used to evaluate the slowdown of some primitive operating system operations. The results are shown in Table 3. CloudVisor does not cause much slowdown for primitives that do not trap to the VMM, like *stat*, *mmap* and *Bcopy*. *10k file delete* and *sh proc* incur relatively high slowdown. *10k file delete* tests the rate of file remove per second that involves inode updates in the filesystem. *sh proc* executes a binary that also involves expensive file operations.

8.4 Boot Time and Memory Overhead

As CloudVisor requires to do decryption and integrity checking when booting a guest VM, we also compared the time of completely booting a VM under CloudVisor and Xen. As expected, the VM booting time under CloudVisor suffered a 2.7X slowdown (33.3s vs. 9.1s) when booting up a VM with 1 GB memory. This is due to the frequent privilege instruction emulation, I/O interposition and cryptographic operations during system bootup. We believe that such overhead is worthwhile as CloudVisor ensures tamper-resistant bootstrap of a guest VM.

The major memory usage in CloudVisor is for the storage of non-leaf Merkle tree metadata for each guest VM and a bounce buffer. This counts up to 10 MB per VM. The memory consumption of the rest part of CloudVisor is less than 1 MB. Hence, for commodity server machines with abundant memory, the memory overhead of CloudVisor is negligible.

9. LIMITATION AND FUTURE WORK

CloudVisor is a first attempt to leverage the concept of nested virtualization [23, 13] for the case of TCB reduction and security protection of virtualized multi-tenant cloud. There are still ample improvement spaces over CloudVisor, which will be our further work:

Enhancing Protection: A common problem exists in CloudVisor and other similar systems when trying to protect from a hostile service provider [15, 17, 71, 40], as the service provider (e.g., OS or VMM) might mislead or even refuse to serve a client (e.g., an application or a VM). Specifically, a malicious VMM might try to mislead a VM by even discarding I/O requests from a VM. One possible mitigation technique is to let a VM or CloudVisor to proof-check services by the VMM.

We are also investigating the tradeoff in functionality division between hardware and software. As the functionality of CloudVisor is very simple and fixed, it might be feasible to implement CloudVisor in hardware or firmware (like Loki [72]).

Impact on VMM's Functionality: While CloudVisor is mostly compatible with existing operations in virtualization stack like save and restore, it does inhibit some VM introspection systems [30, 29] that require introspection of a VM's memory, as they can only see encrypted data. Further, CloudVisor follows a strict isolation policy among VMs, which may prevent some memory sharing systems [67, 25, 43] from working. This could be simply enabled by allowing some pages being shared read-only among VMs, and validating any changes to these pages in CloudVisor. Finally, CloudVisor currently uses a fail-stop approach against possible attacks or crashes. This can be replaced by a fail-safe approach to improving the reliability of VMs atop CloudVisor.

VMM Cooperation: Our currently system is designed to work with existing VMMs, to retain backward compatibility. As demonstrated by our optimization on *VM read* and *VM write*, slight changes to the Xen VMM to make it cooperative with CloudVisor may further improve the performance and reduce the complexity of CloudVisor.

Supporting Other VMMs: We currently only tested CloudVisor for the Xen VMM. A part of our further work includes the evaluation of CloudVisor on other VMMs (such as VMware, KVM and BitVisor [57]) and OSes (such as MAC OS and FreeBSD). Further, we will also investigate how the VMM could be adapted to support para-virtualization in CloudVisor.

Verification: CloudVisor is currently built with a very small code base, thus would be possible to formally verify its correctness and security properties [34] and verify its implementation with software model checking [28], which will be our further work.

10. RELATED WORK

Securing the Virtualization Layer: The increasing number of systems relying on trustworthiness of the virtualization layer makes the security of this layer more important than ever before. Hence, there are several efforts in improving or even reconstructing the virtualization layer to increase the security of the virtualization stack. For example, HyperSentry [11] and HyperSafe [68] both target at improving the security of the VMM by either measuring its integrity dynamically or enforcing control-flow integrity. NOVA [60] is micro-kernel based VMM that decouples the traditional monolithic VMM into a component-based system, and improves security by introducing capability-based access control for different components in a VMM. The security of the management software in Xen [12] is also improved by moving the domain (i.e., VM) building utilities into a separate domain [44]. However, these systems aim at protecting the virtualization layer from external attacks to the VM stack, but without considering possible attacks that leverage legal maintenance operations from the cloud operators, which is a new requirement in multi-tenant cloud. Hence, such systems are orthogonal to CloudVisor and could reduce the possibility of compromises in the VMM.

NoHype [31] tries to address the trustworthiness of multi-tenant clouds by removing the virtualization layer during execution. However, removing the virtualization layer may also lose some useful features such as sharing resources across multiple VMs, which are key features of multi-tenant clouds. Further, NoHype still trusts the VM management software and requires changes to existing hardware and virtualization stack and there is no available implementation of such a system. By contrast, CloudVisor is backward-compatible with commercial virtualization software and is with a smaller trusted computing base.

Protecting Application Code: The threat model and goal of CloudVisor are similar to systems that provide protection of individual processes inside an untrustworthy operating system, such as CHAOS [15], Overshadow [17] and SP^3 [71]. Compared to systems providing protection at the process level, protection at the VM level is much simpler and results in a much smaller TCB. This is because the VM interface and abstraction is with less semantics and thus much simpler and cleaner than those at the process level. For example, there are more than 300 and 1000 system calls with rich semantics (e.g., *ioctl*) in Linux and Windows accordingly. Porting the protection mechanism from one OS to another is usually non-trivial. By contrast, the interface between the VM and VMM can mostly be expressed using the *VM exit* and *VM entry* primitives.

To ensure secure execution of specific code (e.g., SSL) in some applications, researchers proposed several systems to ensure code integrity and data secrecy of such code by leveraging trusted computing hardware [39, 38] and virtualization [40]. Compared to these systems that protect only a part of application software, CloudVisor provides protection at the whole VM level, which naturally fits with the context of multi-tenant cloud.

Virtualization-based Attacks and Defenses: On the positive side, virtualization provides a new playground for system security. Many prior literatures use special-purpose VMMs to improve security of operating systems [53, 69, 48], or extend existing VMMs with security-enhanced policies or mechanisms [30, 29] Compared to CloudVisor, these systems only protect a part of a program, while CloudVisor aims at protecting the entire virtual machine. The above systems could be incorporated with CloudVisor to prevent attacks from the network side, which may form a more secure cloud platform.

On the negative side, virtualization has also been used as a means to mount attack traditional operating systems and virtualization system [32, 50, 51]. When it is used to attack a VMM [51], the rootkit also needs to implement part of the nested virtualization to give the illusion that the VMM is running on bare metal.

Software and Hardware Support for Trusted Computing: Building more trustworthy software stack and platforms is always the concerns of researchers. The trusted computing groups have proposed the *Trusted Platform Module* [66] for the purpose of measuring a platform [52]. There is also several research on software-based remote attestation (e.g., Pioneer [54] and SWATT [55]). Such attestation techniques could be integrated into CloudVisor for remote attestation of code inside leased VMs to prevent from network-side attacks.

Machine partitioning using virtualization (e.g., Terra [21], NGSCB [47]) tries to satisfy the security requirements of diverse applications by providing different types of close-box and open-box VMs to applications. However, no defense against operators is provided in these systems.

There are also many architectural proposals that aim at providing security protection to applications. For examples, many architectural enhancements [36, 61] have been proposed to support trusted execution of an application within an untrusted operating system. System designers also leverage such support to build operating systems (e.g., XOMOS [37]).

Nested Virtualization: Researchers have investigated integrating nested virtualization into commodity VMMs, which forms an even larger TCB. For example, the recent Turtles project [13] investigates the design and implementation of nested virtualization to support multi-level virtualization in KVM [33]. In contrast, CloudVisor leverages nested virtualization to minimize TCB by separating the functionality for nested virtualization from the functionality for resource management, and further enhances the nested VMM with security protection of the hosted virtual machines.

11. CONCLUSION

Current multi-tenant cloud faces two major sources of threats: attacks to the virtualized infrastructure by exploiting possible security vulnerabilities in the relative large and complex virtualized software stack; and attacks originated from stealthy accesses to sensitive data from cloud operators. This paper presented a lightweight approach that introduces a tiny security monitor underneath the VMM to defend against these attacks. Our system, called CloudVisor, provides strong privacy and integrity guarantees even if the VMM and the management software are in control by adversaries. CloudVisor achieved this by exploiting commercially-available hardware support for virtualization and trusted computing. Performance evaluation showed that CloudVisor incurred moderate slowdown for I/O intensive applications and very small slowdown for other applications.

12. ACKNOWLEDGMENTS

We thank our shepherd Nickolai Zeldovich and the anonymous reviewers for their insightful comments. Cheng Tan, Yubin Xia and Rong Chen helped to prepare the final version of this paper. This work was funded by China National Natural Science Foundation under grant numbered 90818015 and 61003002, a grant from the Science and Technology Commission of Shanghai Municipality numbered 10511500100, Fundamental Research Funds for the Central Universities in China and Shanghai Leading Academic Discipline Project (Project Number: B114).

13. REFERENCES

[1] Bitlocker drive encryption technical overview. `http://technet.microsoft.com/en-us/library/cc766200%28WS.10%29.aspx`.

[2] Common vulnerabilities and exposures. `http://cve.mitre.org/`.

[3] Filevault in mac osx. http://www.apple.com/macosx/whats-new/features.html#filevault2.

[4] Intel advanced encryption standard instructions (aes-ni). `http://software.intel.com/en-us/articles/intel-advanced-encryption-standard-instructions-aes-ni/`, 2010.

[5] D. Abramson, J. Jackson, S. Muthrasanallur, G. Neiger, G. Regnier, R. Sankaran, I. Schoinas, R. Uhlig, B. Vembu, and J. Wiegert. Intel virtualization technology for directed I/O. *Intel technology journal*, 10(3):179–192, 2006.

[6] Amazon Inc. Amazon Elastic Compute Cloud (Amazon EC2). `http://aws.amazon.com/ec2/`, 2011.

[7] Amazon Inc. Amazon machine image. `http://aws.amazon.com/amis`, 2011.

[8] Amazon Inc. Amazon web service customer agreement. `http://aws.amazon.com/agreement/`, 2011.

[9] AMD Inc. Secure virtual machine architecture reference manual, 2005.

[10] Apache. ab - apache http server benchmarking tool. `http://httpd.apache.org/docs/2.0/programs/ab.html`, 2011.

[11] A. Azab, P. Ning, Z. Wang, X. Jiang, X. Zhang, and N. Skalsky. HyperSentry: enabling stealthy in-context measurement of hypervisor integrity. In *Proc. CCS*, pages 38–49, 2010.

[12] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proc. SOSP*. ACM, 2003.

[13] M. Ben-Yehuda, M. D. Day, Z. Dubitzky, M. Factor, N. Har'El, A. Gordon, A. Liguori, O. Wasserman, and B.-A. Yassour. The turtles project: Design and implementation of nested virtualization. In *Proc. OSDI*, 2010.

[14] H. Chen, J. Chen, W. Mao, and F. Yan. Daonity-grid security from two levels of virtualization. *Information Security Technical Report*, 12(3):123–138, 2007.

[15] H. Chen, F. Zhang, C. Chen, Z. Yang, R. Chen, B. Zang, P. Yew, and W. Mao. Tamper-resistant execution in an untrusted operating system using a virtual machine monitor. *Parallel Processing Institute Technical Report, Number: FDUPPITR-2007-0801, Fudan University*, 2007.

[16] P. Chen and B. Noble. When virtual is better than real. In *Proc. HotOS*, 2001.

[17] X. Chen, T. Garfinkel, E. Lewis, P. Subrahmanyam, C. Waldspurger, D. Boneh, J. Dwoskin, and D. Ports. Overshadow: a virtualization-based approach to retrofitting protection in commodity operating systems. In *Proc. ASPLOS*, pages 2–13. ACM, 2008.

[18] Y. Dong, Z. Yu, and G. Rose. SR-IOV networking in Xen: Architecture, design and implementation. In *Proc. Workshop on I/O virtualization*. USENIX, 2008.

[19] B. Fitzpatrick. Distributed caching with memcached. *Linux journal*, 2004.

[20] Flexiant Inc. Flexiscale public cloud. `http://www.flexiant.com/products/flexiscale/`.

[21] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: A virtual machine-based platform for trusted computing. *ACM SIGOPS Operating Systems Review*, 37(5):206, 2003.

[22] T. Garfinkel and M. Rosenblum. When virtual is harder than real: Security challenges in virtual machine based computing environments. In *Proc. HotOS*, 2005.

[23] R. Goldberg. Architecture of virtual machines. In *Proceedings of the workshop on virtual computer systems*, pages 74–112, 1973.

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

[25] D. Gupta, S. Lee, M. Vrable, S. Savage, A. C. Snoeren, G. Varghese, G. M. Voelker, and A. Vahdat. Difference engine: harnessing memory redundancy in virtual machines. In *Proc. OSDI*, pages 309–322, 2008.

[26] J. Heiser and M. Nicolett. Assessing the security risks of cloud computing. http://www.gartner.com/DisplayDocument?id=685308, 2008.

[27] Intel Inc. Intel trusted execution technology. www.intel.com/technology/security/, 2010.

[28] R. Jhala and R. Majumdar. Software model checking. *ACM Computing Surveys (CSUR)*, 41(4):1–54, 2009.

[29] X. Jiang and X. Wang. Out-of-the-box monitoring of VM-based high-interaction honeypots. In *Proc. RAID*, pages 198–218, 2007.

[30] X. Jiang, X. Wang, and D. Xu. Stealthy malware detection through vmm-based out-of-the-box semantic view reconstruction. In *Proc. CCS*, pages 128–138. ACM, 2007.

[31] E. Keller, J. Szefer, J. Rexford, and R. Lee. NoHype: virtualized cloud infrastructure without the virtualization. In *Proc. ISCA*, pages 350–361, 2010.

[32] S. King, P. Chen, Y. Wang, C. Verbowski, H. Wang, and J. Lorch. SubVirt: Implementing malware with virtual machines. In *Proc. S&P (Oakland)*, 2006.

[33] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. kvm: the linux virtual machine monitor. In *Linux Symposium*, 2007.

[34] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, et al. seL4: Formal verification of an OS kernel. In *Proc. SOSP*, pages 207–220, 2009.

[35] M. Krohn, P. Efstathopoulos, C. Frey, F. Kaashoek, E. Kohler, D. Mazieres, R. Morris, M. Osborne, S. VanDeBogart, and D. Ziegler. Make least privilege a right (not a privilege). In *Proc. HotOS*, 2005.

[36] D. Lie, C. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J. Mitchell, and M. Horowitz. Architectural support for copy and tamper resistant software. In *Proc. ASPLOS*, pages 168–177, 2000.

[37] D. Lie, C. A. Thekkath, and M. Horowitz. Implementing an untrusted operating system on trusted hardware. In *Proc. SOSP*, pages 178–192, 2003.

[38] J. McCune, B. Parno, A. Perrig, M. Reiter, and H. Isozaki. Flicker: An execution infrastructure for TCB minimization. In *Proc. Eurosys*, pages 315–328, 2008.

[39] J. McCune, B. Parno, A. Perrig, M. Reiter, and A. Seshadri. Minimal TCB code execution. In *Proc. S&P (Oakland)*, pages 267–272, 2007.

[40] J. M. McCune, Y. Li, N. Qu, Z. Zhou, A. Datta, V. Gligor, and A. Perrig. TrustVisor: Efficient TCB Reduction and Attestation. In *Proc. S&P (Oakland)*, 2010.

[41] L. McVoy and C. Staelin. lmbench: Portable tools for performance analysis. In *Proc. Usenix ATC*, 1996.

[42] R. Merkle. Protocols for public key cryptosystems. In *Proc. S&P (Oakland)*, 1980.

[43] G. Miłós, D. G. Murray, S. Hand, and M. A. Fetterman. Satori: enlightened page sharing. In *Proc. Usenix ATC*, 2009.

[44] D. Murray, G. Milos, and S. Hand. Improving Xen security through disaggregation. In *Proc. VEE*, pages 151–160, 2008.

[45] G. Neiger, A. Santoni, F. Leung, D. Rodgers, and R. Uhlig. Intel virtualization technology: Hardware support for efficient processor virtualization. *Intel Technology Journal*, 10(3):167–177, 2006.

[46] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. The eucalyptus open-source cloud-computing system. In *Proc. CCGRID*, pages 124–131, 2009.

[47] M. Peinado, Y. Chen, P. England, and J. Manferdelli. NGSCB: A trusted open system. In *Information Security and Privacy*, pages 86–97, 2004.

[48] R. Riley, X. Jiang, and D. Xu. Guest-transparent prevention of kernel rootkits with vmm-based memory shadowing. In *Proc. RAID*, pages 1–20, 2008.

[49] 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 *Proc. CCS*, pages 199–212. ACM, 2009.

[50] J. Rutkowska. Introducing Blue Pill. *The official blog of the invisiblethings. org. June*, 22, 2006.

[51] J. Rutkowska and A. Tereshkin. Bluepilling the Xen Hypervisor. *Black Hat USA*, 2008.

[52] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn. Design and implementation of a tcg-based integrity measurement architecture. In *Proc. USENIX Security*, 2004.

[53] A. Seshadri, M. Luk, N. Qu, and A. Perrig. SecVisor: A tiny hypervisor to provide lifetime kernel code integrity for commodity OSes. In *Proc. SOSP*, 2007.

[54] A. Seshadri, M. Luk, E. Shi, A. Perrig, L. Van Doorn, and P. Khosla. Pioneer: Verifying integrity and guaranteeing execution of code on legacy platforms. In *Proc. SOSP*, pages 1–16, 2005.

[55] A. Seshadri, A. Perrig, L. Van Doorn, and P. Khosla. SWATT: Software-based attestation for embedded devices. In *Proc. S&P (Oakland)*, pages 272–282, 2004.

[56] J. Shi, X. Song, H. Chen, and B. Zang. Limiting cache-based side-channel in multi-tenant cloud using dynamic page. In *Proc. HotDep*, 2011.

[57] T. Shinagawa, H. Eiraku, K. Tanimoto, K. Omote, S. Hasegawa, T. Horie, M. Hirano, K. Kourai, Y. Oyama, E. Kawai, et al. BitVisor: a thin hypervisor for enforcing i/o device security. In *Proc. VEE*, pages 121–130. ACM, 2009.

[58] L. Singaravelu, C. Pu, H. H
"artig, and C. Helmuth. Reducing TCB complexity for security-sensitive applications: Three case studies. In *Proc. Eurosys*, 2006.

[59] SPEC. Specjbb 2005. http://www.spec.org/jbb2005/, 2005.

[60] U. Steinberg and B. Kauer. NOVA: A microhypervisor-based secure virtualization architecture. In *Proc. Eurosys*, pages 209–222. ACM, 2010.

[61] G. Suh, D. Clarke, B. Gassend, M. Van Dijk, and S. Devadas. AEGIS: architecture for tamper-evident and tamper-resistant processing. In *Proc. Supercomputing*, 2003.

[62] T. R. Team. Rackspace cloud. http://www.rackspacecloud.com/.

[63] TechSpot News. Google fired employees for breaching user privacy. http://www.techspot.com/news/40280-google-fired-employees-for-breaching-user-privacy.html, 2010.

[64] The Nimbus Team. Nimbus project. http://www.nimbusproject.org/.

[65] A. Tridgell. Dbench filesystem benchmark. http://samba.org/ftp/tridge/dbench/.

[66] Trusted Computing Group. Trusted platform module. http://www.trustedcomputinggroup.org/, 2010.

[67] C. A. Waldspurger. Memory resource management in vmware esx server. In *Proc. OSDI*, pages 181–194, 2002.

[68] Z. Wang and X. Jiang. HyperSafe: A lightweight approach to provide lifetime hypervisor control-flow integrity. In *Proc. S&P (Oakland)*, pages 380–395, 2010.

[69] Z. Wang, X. Jiang, W. Cui, and P. Ning. Countering kernel rootkits with lightweight hook protection. In *Proc. CCS*, pages 545–554. ACM, 2009.

[70] C. Weinhold and H. Härtig. jVPFS: Adding Robustness to a Secure Stacked File System with Untrusted Local Storage Components. In *Proc. Usenix ATC*, 2011.

[71] J. Yang and K. G. Shin. Using hypervisor to provide data secrecy for user applications on a per-page basis. In *Proc. VEE*, pages 71–80, 2008.

[72] N. Zeldovich, H. Kannan, M. Dalton, and C. Kozyrakis. Hardware enforcement of application security policies using tagged memory. In *Proc. OSDI*, pages 225–240, 2008.