Improving Virtualization Security by Splitting Hypervisor into Smaller Components

Wuqiong Pan[1,2,⋆], Yulong Zhang[2], Meng Yu[2], and Jiwu Jing[1]

[1] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
{wqpan,jing}@lois.cn
[2] Department of Computer Science, Virginia Commonwealth University, Richmond, VA, 23284 USA
{wpan,zhangy44,myu}@vcu.edu

Abstract. In cloud computing, the security of infrastructure is determined by hypervisor (or Virtual Machine Monitor, VMM) designs. Unfortunately, in recent years, many attacks have been developed to compromise the hypervisor, taking over all virtual machines running above the hypervisor. Due to the functions a hypervisor provides, it is very hard to reduce its size. Including a big hypervisor in the Trusted Computing Base (TCB) is not acceptable for a secure system design. Several secure, small, and innovative hypervisor designs, e.g., TrustVisor, CloudVisor, etc., have been proposed to solve the problem. However, these designs either have reduced functionalities or pose strong restrictions to the virtual machines. In this paper, we propose an innovative hypervisor design that splits hypervisor's functions into a small enough component in the TCB, and other components to provide full functionalities. Our design can significantly reduce the TCB size without sacrificing functionalities. Our experiments also show acceptable costs of our design.

Keywords: VMM, Hypervisor, Cloud computing, TCB.

1 Introduction

Virtualization techniques allow multiple operating systems (OSs) to run concurrently on a host computer. By sharing hardware, resource utilization can greatly be improved. Virtualization is also the key technology of cloud computing. Some software, such as Xen [1], can provide hardware virtualization by adding a new software layer called hypervisor beneath all Virtual Machines (VMs). A hypervisor emulates independent hardware resources for every VM. Both Intel and AMD have developed new extensions [2,3] for hardware based virtualization, which can simplify hypervisor designs.

In cloud computing, legal users and attackers may share the same physical server. Thus, it is important to isolate VMs from each other. In a virtualization

⋆ This work was done while the first author worked at Virginia Commonwealth University.

N. Cuppens-Boulahia et al. (Eds.): DBSec 2012, LNCS 7371, pp. 298–313, 2012.
© IFIP International Federation for Information Processing 2012

architecture, the hypervisor is responsible for isolation. In current hypervisor designs, it also emulates the hardware and provides other security functions for VMs. Many researchers argued that the current hypervisor designs include too many functions [4,5]. For an example, Xen has about 270k lines of codes (LOC), which is difficulty to implement without bugs. Unfortunately, many vulnerabilities in hypervisor have already been discovered [6,7,8,9,10,11].

To address the problem, a lot of efforts [4,5,12,13,14,15,16] have been made to improve the isolation. Among these work, Overshadow [12] provides fine-grained isolation which protects applications from a compromised OS. SecureME [13] has the same function and can defend against hardware attacks by using a secure processor substrate. Bastion [15] defines a struct of module and protects it from hardware attacks. Overshadow, SecureME and Briston can provide fine-gained isolation for applications or modules. These work add new functions to the hypervisor. As a result, the size of the hypervisor is increased. Therefore, the added codes increase the size of TCB and they may introduce new vulnerabilities as well.

In contrast, some efforts try to reduce the number of functions provided by a hypervisor to make the hypervisor, thus TCB, smaller. For examples, SICE [14] provides hardware isolation for one workload on a core by using the System Management Mode (SMM) of x86 processors. NoHype [4,5] eliminates the hypervisor attack surface by removing the hypervisor after booting the guest VMs. NoHype pre-allocate resources for a VM before the boot procedure. CloudVisor [16] can protect VMs from a compromised hypervisor by adding an additional hypervisor layer. These new designs also introduce some restrictions. For examples, both SICE and Nohype have the limit of one protected workload, or VM per core on multi-core processors. Also, NoHype does not support dynamic resources allocation. CloudVisor does not allow hypervisor to access VM pages and it becomes a big burden for CloudVisor to determine which pages of the guest VMs can be accessed by the master hypervisor.

Moreover, the security problem of shared management domain (or hypervisor in CloudVisor) are not completely eliminated in afore mentioned designs. For examples, in NoHype design, the attackers can attack the management VM. Although the management OS may be well configured by cloud provider, an OS is usually considered more vulnerable to attacks than a hypervisor. In CloudVisor design, a new hypervisor is used to intercept all communications between the master hypervisor and VMs. Even though the attack surface is smaller, once the master hypervisor is compromised, all VMs can still be affected.

In this paper, we propose a split hypervisor architecture, called *SplitVisor*, which has a small TCB and does not limit the functions of a hypervisor. Our architecture leverages nested virtualization [17,18]. In our design, every VM has its own hypervisor, called *GuestVisor*. Users can customize the GuestVisor. A SplitVisor is underneath all GuestVisors and VMs. The SplitVisor is responsible for isolation, which is the only part in TCB.

Table 1 shows the comparison of the proposed architecture and existing ones. SplitVisor, CloudVisor and SICE are the only ones which have a small TCB. All

of them sacrifice some of hypervisor functions except SplitVisor. SplitVisor is the only one which has a stable TCB, which means that the TCB need little or even none of change when adding new functions to hypervisor. All others need to modify the TCB if they add new functions to the hypervisor. A stable TCB can greatly reduce the verification costs when upgrading secure software. SplitVisor, CloudVisor, and SICE can defend against attacks from a compromised hypervisor. The security of SICE is ensured by the SMM mode of x86 processors while the security of SplitVisor and CloudVisor are ensured by a nested architecture. CloudVisor can block a hypervisor from accessing its VMs' data. The protected units of these work are also different. Only SplitVisor, CloudVisor, and Nohype have a VM as the unit of protection. Other work either have applications [12,13], or hardware [19,20,21,22], as the unit of protection, which are out of the scope of this paper. These protections are still compatible with SplitVisor design.

Table 1. Comparison of different designs

	TCB Stability[1]	**Functions**	**Hypervisor**[2]	**VM**[3]	**Protection**	**Assertion**	**HD**[4]
SplitVisor	small √	full	√	√	VM	√	
CloudVisor [16]	small	partial	√		VM	√	
SICE [14]	small	partial	√	√	region	√	
NoHype [4,5]	large	partial			VM		
Overshadow [12]	large	full			application		
Bastion [15]	large	full			module		√
SecureME [13]	large	full			application		√
XOM [19,20]	large	full			region		√
AEGIS [21]	large	full			region		√
AISE [22]	large	full			region		√

The main contributions of our work include:

- A small and stable TCB. Both GuestVisors and the management OS are not in the TCB. They cannot access other VMs. Most of functions can be added to GuestVisors without modifying the TCB.
- Supporting full-functions. Unlike other architectures which have a small TCB, SplitVisor does not eliminate any functions from hypervisors.
- Allowing users to verify the execution environment. Users can get the assertion of the environment.

The rest of this paper is organized as follows. Section 2 discusses our goals and shows the whole architecture of SplitVisor. Section 3 describes SplitVisor in booting, memory management, scheduling and some other details. Section 4 compares SplitVisor with other recent work. Section 5 shows the related work in isolation. Section 6 gives an conclusion of the paper.

[1] A TCB design is *stable* if the TCB needs little or even none of changes when adding new functions to the hypervisor.
[2] Attacks can be confined to the hypervisor when the hypervisor is compromised.
[3] VMs are protected when the hypervisor is compromised.
[4] Hardware level attacks, such as memory tapping [23], can be handled.

2 Overview

2.1 Design Principles

Cloud services are usually provided using virtualization techniques. The examples are Amazon EC2 [24], Eucalyptus [25], FlexiScale [26], Nimbus [27], and RackSpace [28]. In order to attract more cloud users, they continuously improve their products by adding more and more functions to the platform. As the result, the size of TCB, the hypervisor, also increases prominently. Table 2 shows the TCB size of Xen from version 2.0 to version 4.0 [1].

Table 2. Xen TCB size (by Lines of Code)

	Hypervisor	**Kernel of Domain 0**	**Tools**	**TCB**
Xen 2.0	45k	4,136k	26k	4,027k
Xen 3.0	121k	4,807k	143k	5,071k
Xen 4.0	270k	7,560k	647k	8,477k

In the table, the size of Xen 4.0 hypervisor is six times as large as that of Xen 2.0. Although Xen 2.0 is more secure in term of hypervisor size, Xen 4.0 is more attractive to users because of the new features, such as Xen access control, I/O optimization, and Memory page sharing. Domain 0 and tools are also parts of the TCB in Xen in addition to the hypervisor, because they can access the data of all VMs. For an example, the `xm` tool can dump the memory of a VM.

A possible way to reduce the size of TCB is to remove Domain 0 and Xen tools from the TCB, which requires disabling some functions, or encrypting data to prevent accesses from Domain 0. However, this will limit the functionalities of Xen. Cloud providers usually try to provide as many functions as them can. But for a particular user, he may only need a small set of the functions. In SplitVisor, every user can choose a *GuestVisor* which serves as the current hypervisor. The user can choose a hypervisor with the necessary functions, then he will not suffer vulnerabilities of the unneeded functions. Although the GuestVisor still can be attacked by its own VM, the attack will not affect other GuestVisors and VMs. The isolation is ensured by the SplitVisor. Thus, the SplitVisor should have a small code base to ensure its security.

2.2 Assumptions

Our assumptions are described as follows.

Adversaries: We assume that the attackers can easily control a VM. For an example, the attackers can buy a VM. The attackers can invade a management OS and a rich-functions hypervisor. They can send any instructions in the name of the hypervisor. The attackers can also sniff I/O and steal users' OS images. We assume that the attackers cannot physically access the machines. The data

centers are usually protected by well-trained guards. The cloud providers have no motivation to use malicious hardware. Using malicious hardware easily leaves evidence, which definitely ruins cloud providers' reputation. The cloud providers are usually famous companies, while cloud clients are usually small companies. They are not competitors in most cases.

Security Guarantees: The main goal of SplitVisor is to provide isolated environments for VMs. SplitVisor directly provides CPU isolation and memory isolation for VMs. Other security attributes can be achieved based on these two attributes. SplitVisor allows users to verify its TCB. Users can get evidence that their OSs run in the expected environment.

Non-guaranteed Goals: SplitVisor cannot guarantee the availability of a particular VM. If a VM cannot get CPU time slices from the management software, the VM will be blocked, but this is easily discovered by outside.

SplitVisor is not designed to defend against side-channel attacks. Xen provides Chinese Wall (CHWALL) policy, which can control the set of VMs on the same machine. Well-defined CHWALL policy can reduce side-channel attacks [29]. Besides hypervisor level policies, some applications also have built-in mechanisms to reduce side-channel attacks [30].

2.3 Architecture

Both Intel and AMD have added new extensions to support the hypervisor layer, like Intel's Virtualization Technology for x86 (VT-x) and AMD's Secure Virtual Machine (SVM). With the new extensions, the Intel processors have two operation modes: virtual-machine extension (VMX) root mode and VMX non-root mode. In general, a hypervisor will run in VMX root mode and a guest OS will run in VMX non-root mode. The control information of VMX transition is stored in a data structure called virtual-machine structure (VMCS). A VMCS includes almost all environment parameters of a guest OS, such as registers.

SplitVisor has two types of components that belong to one hypervisor before: GuestVisors and a SplitVisor, as shown in Fig 1. The SplitVisor runs in VMX root mode, while both the GuestVisors and VMs run in VMX non-root mode. Both GuestVisors and VMs have VMCSs. A GuestVisor's VMCS is controlled by the SplitVisor, and a VM's VMCS is controlled by a GuestVisor. Most requests from a guest OS are handled by the GuestVisor under it. A GuestVisor is responsible for the execution environment of a guest OS. If a user wants to add some functions into the current hypervisor layer, he can add them into the GuestVisor. The isolation among all GuestVisors and VMs is ensured by the SplitVisor. All VMX transition instructions are executed in the SplitVisor. The SplitVisor is also responsible for memory isolation.

The features of the SplitVisor, a GuestVisor and a VM are shown in Table 3. The SplitVisor does not provide a complete virtualized running environment to a GuestVisor. A GuestVisor knows that it is not running on the bare hardware. A VM runs in a virtualized environment emulated by a GuestVisor. A VM can be para-virtualized or full-virtualized.

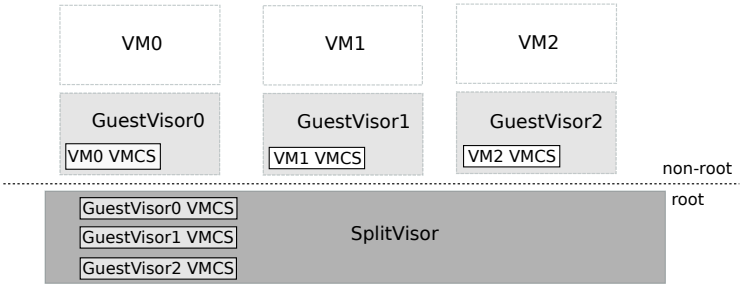

Fig. 1. SplitVisor architecture

Table 3. Units in SplitVisor

Unit	Functions	Transparency
SplitVisor	VMX transition, memory isolation	-
GuestVisor	OS execution environment, extra functions	Not support
VM	Running OS	Both para- and full- virtualization

Security Analysis: If a GuestVisor is compromised, the SplitVisor will ensure other GuestVisors and VMs are not affected. In other words, the isolation is ensured. But the VM of the compromised GuestVisor is not protected.

3 Design Details

3.1 Secure Boot

The boot process of a computer is the first step to set up a secure environment for guest OS. In our design, the only trusted component is the SplitVisor. The secure boot process is described as follows.

First, *verify the SplitVisor*. Users can verify the SplitVisor, which is supported by a Trusted Platform Module (TPM) [31]. A TPM is an secure chip which can help to protect the integrity of the boot process and the SplitVisor. Users can also obtain the SplitVisor's public key with the help of the TPM.

Second, *make a new image*. The binary codes of both user's VM and GuestVisor should be provided by users. Users must encrypt all or part of the codes with a symmetric key. The symmetric key should be encrypted by the SplitVisor's public key, so the SplitVisor can decrypt it. All the plain text should be signed to protect against unauthorized modification. The total data sent to cloud server includes the following: encrypted image, encrypted symmetric key, plain data, signature over everything, and the public key certificate of the user (signer).

Third, *prepare the environment of a GuestVisor*. When the SplitVisor receives the above data from users. It will decrypt all data and verify the signature. If the verification goes through, the SplitVisor will set up the running environment for

a GuestVisor. The allocation of memory and CPU is decided by the management software. The boot process of VM is controlled by the GuestVisor.

Fourth, *authentication between the user and the guest OS*. In traditional login, only OS verifies users. In cloud computing, we need two-way authentication, because attackers can run a malicious system, and try to trick user to login. Users can put the key in guest OS before sending it to cloud providers. When guest OS boots up in remote server, they can authenticate each other based on the key. All communication after that also should be protected by the key.

Boot Process of the Management VM: The management VM is the one we should boot first. The boot process is a little different at the first step and the second step. Cloud providers should get the public key of the TPM through some public channels. The making of the management VM's image is the same as that of others. However, the image must be stored in a disk, or other storage devices which can be easily read by a SplitVisor. All the memory except the SplitVisor's, CPUs and devices are belonged to the management VM at first. It will assign the resources to other VMs in the future execution.

Security Analysis: Without the authentication key, a malicious OS cannot cheat users. Attackers cannot get the key which is encrypted in the second step. If attackers do not crack the key but modify the image, it will be detected when verifying signatures in the third step.

3.2 Memory Management

Both Intel and AMD have extended two layer address translation to three layer address translation, called Extended Page-Table (EPT) and nested paging. The page table (PT) specified by CR3 is still responsible for translating virtual frame number (VFN) to physical frame number (PFN). A new table called EPT for Intel is responsible for translating PFN to machine frame number (MFN). The address of EPT is specified by EPTP, a VMCS field. The three layer model is shown in Fig 2 to compare our SplitVisor design and the current hypervisor design.

The SplitVisor will set up a default EPT for a GuestVisor. The SplitVisor assigns all allowed memory page to the GuestVisor at the beginning. The range of PFN is fixed, for an example, it always starts from 0. For further processing, SplitVisor will allow GuestVisor to read its own EPT.

After a GuestVisor is booted, it will prepare the environment for a VM. Firstly, GuestVisor sets up an EPT for the VM. The GuestVisor can get the addresses of its MFNs by reading its own EPT. It can keep some MFNs for its own use, and assign others to VM. Finally, GuestVisor write the address of VM's EPT to VM's VMCS.

When a VM is booted, SplitVisor will check its EPT to see if all the MFNs are from the GuestVisor. So a user cannot access others' MFN by booting VM. After the checking, the SplitVisor marks the EPT page as read-only.

A GuestVisor has two means to manipulate the address translation of a VM. One is to modify the VM's EPT, another is to modify the VM's PT. Modifying the PT is the same as shadow paging. Modifying the EPT is the same as that we

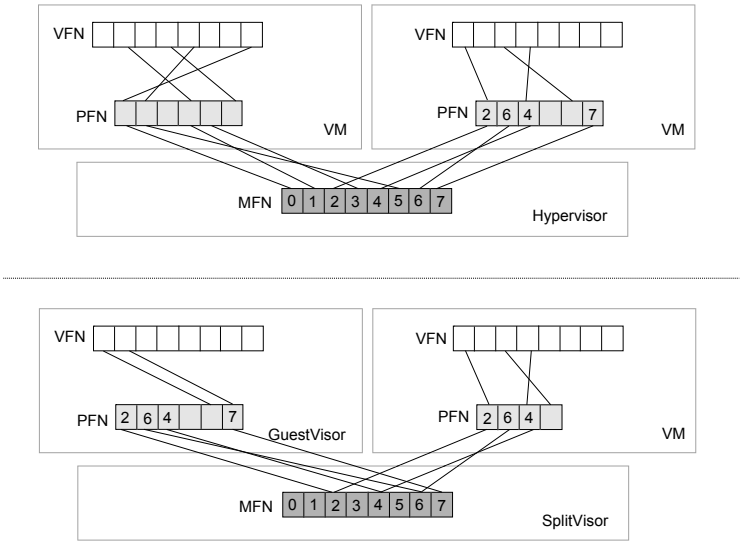

Fig. 2. Comparison of the current hypervisor design (top) and SplitVisor design (bottom) in terms of address translation

support EPT functions between a GuestVisor and a VM. The only difference is that a VM's EPT is marked as read-only by the SplitVisor. Any modification of the VM's EPT will cause a VM exit. The SplitVisor will check the modification and do it for the GuestVisor.

The SplitVisor allows a GuestVisor to transfer its memory to other GuestVisors, but the request must be originated by the GuestVisor who owns the memory. The SplitVisor un-maps the pages from the GuestVisor's and VM's EPT, and maps them to another GuestVisor. Then the SplitVisor adds all the pages to the EPT of the second GuestVisor, and the second GuestVisor can decide how to use it. Memory sharing is similar to memory transferring except some flags of the memory pages are different.

There is a trade off about whether the management software should be allowed to forcibly receive other VMs' memory. If it is allowed, the SplitVisor can reset the received pages to zero when they are remapped. But the management software can still observe the action of a GuestVisor when different pages are received. It will introduce a way to attack the GuestVisor and VMs. Therefore, our SplitVisor design does not allow it. In this situation, the GuestVisor may do not want to give up its memory even if it does not need it. We can charge more to the GuestVisor's owner, it is fair since the user pays more when using more resources. A problem of this strategy is that we cannot receive GuestVisor's memory when it is crashed. It is difficulty to know the crash for the SplitVisor. The SplitVisor can try to migrate the GuestVisor. If the GuestVisor does not respond , it is crashed. Then SplitVisor can reallocate its memory.

Table 4 shows the comparison of memory management in different designs. In CloudVisor design, VM management is implemented in a modified hypervisor above the bottom layer called CloudVisor. In SplitVisor design, VM management, memory transferring and memory sharing are controlled by GuestVisors. Thus, the GuestVisors have corresponding privileges to manage VMs. The only work of the SplitVisor is to check the page access permissions. Xen implements all functions in the hypervisor. If Xen wants to add new functions, it has to modify the hypervisor.

Table 4. Comparison of Memory Designs - Where The Functions Are Implemented

Function	**SplitVisor**	**CloudVisor**	**Xen**
Memory isolation	SplitVisor	CloudVisor	Hypervisor
VM management	GuestVisor	Hypervisor	Hypervisor
Memory transferring, sharing	GuestVisor	CloudVisor	Hypervisor

3.3 Scheduling and VMCS

The transition from the root mode to the non-root mode is called *VM entry*, while the opposite is called *VM exit*. The transitions are controlled by a VMCS. The SplitVisor creates a default VMCS for a GuestVisor and controls the scheduling of GuestVisors. A GuestVisor controls the scheduling of the VM running above it. For an example, A GuestVisor may provide many VCPUs to the VM for special purpose. All the VCPUs are managed by the GuestVisor.

Every time a GuestVisor gets a time slice, the SplitVisor will switch into the GuestVisor first. The GuestVisor decides on which VCPU to start and switches back to the SplitVisor, then the SplitVisor switches into the VM. Before switching into the VM, the SplitVisor must modify the VM' VMCS. Some fields in the VMCS should be rewritten by the SplitVisor, such as host registers, which determine the CPU status when returning from the VM. The rewriting should be done every time entering the VM, because the GuestVisor may modify the VMCS.

The SplitVisor does not handle VM exits of a VM. It transfers all VM exits to the GuestVisor except timer and some I/O interrupts. Operations causing VM exits are specified by a VMCS. A GuestVisor can configure VM exits by modifying a VM's VMCS. For an example, if a GuestVisor wants to intercept VM's system calls, it just modifies control filed of the `80h` interrupt.

Table 5 shows the comparison of different scheduling designs. VCPU management is implemented in the hypervisor of CloudVisor, which gives the hypervisor opportunities to attack VMs. Xen implements all functions in the hypervisor. A problem of scheduling is who can get the next time slice. If a scheduling algorithm is implemented in the management software, the attackers can deny the service of a VM if they controls the management software. If the scheduling

algorithm is implemented in the SplitVisor, its parameters cannot be changed dynamically. Xen implements the algorithm in the hypervisor, but the parameters can be modified in the management software. The security level is the same as the one implemented in the management software. The implementation in SplitVisor can be determined case by case. For an example, in Amazon EC2 the time sharing of VMs is predefined. For such situations, the scheduling algorithm can be implemented in the SplitVisor. Otherwise, it can be implemented in management software.

Table 5. Comparison of Scheduling Designs - Where The Functions Are Implemented

Function	**SplitVisor**	**CloudVisor**	**Xen**
VCPUs management	GuestVisor	Hypervisor	Hypervisor
(GuestVisor) hypervisor's VMCS	SplitVisor	CloudVisor	-
VM's VMCS	GuestVisor	Hypervisor	Hypervisor

3.4 Interrupt and Device Management

In order to meet the needs of cloud computing, some device manufactures added virtualization support to their products, as mentioned in [4]. If a device can support virtualization by itself, every VM can get its own devices. For such situations, the SplitVisor can simply distribute the interrupts and I/O ports to each GuestVisor.

If devices do not support virtualization, all VMs have to share the devices. It can be implemented by front-backend drivers. The SplitVisor assign the devices to one GuestVisor. All request are handled by the special GuestVisor. From the SplitVisor's perspective, some GuestVisors have shared memory with the special GuestVisor, and the SplitVisor does not need to emulate the devices.

3.5 Functions of a GuestVisor

Most functions of the current hypervisors are implemented in the GuestVisor. The GuestVisor is the key to reduce the size of TCB. We list the possible functions of GuestVisor in this section. Different versions of GuestVisors, from light weight ones to the ones with full functionalities, can be provided to the user as options.

I/O Encryption. The GuestVisor that controls devices may be controlled by attackers. The attackers can sniff the I/O data which pass through the devices. Some applications already have built-in mechanisms to protected I/O, such as Bitlocker and SSL. It is hard to decide whether I/O protection should be implemented in the hypervisor. It is more secure to be implemented in the hypervisor, while more flexible in the application. The SplitVisor leaves the decisions to

users. If users want to implement I/O protection in the hypervisor, they can add the new function in the GuestVisor.

VM Life-Cycle Management. Some VM life-cycle functions, such as shutting down a VM, should be implemented in the GuestVisor. When a VM is shutting down, the GuestVisor needs to follow the boot preparation procedure as described in Section 3.1. Also, the GuestVisor writes all data to disk or send them to the management software. The same thing should be done when taking a snapshot of a VM. The management of snapshots is also the GuestVisor's responsibility.

Another important work of the GuestVisor is VM migration. The GuestVisor verifies the new server and creates a new VM image on the target server. All the steps described in secure boot in Section 3.1 should be done again.

Privileged Instructions. Privileged instructions cannot be executed in a VM. In SplitVisor, they are handled by the GuestVisor. If an instruction, e.g., a hyper call, has some arguments in the VM's memory, the GuestVisor can directly read it. In CloudVisor, things are more complex. CloudVisor must fetches page table entries and the arguments for the hypervisor, in the meanwhile the CloudVisor must make sure that no sensitive information is leaked to the hypervisor. The CloudVisor must know the exactly meaning of the instructions. Some work [13] may add new instructions, so it is a big burden to the CloudVisor.

Fine-Gained Isolation. Some architectures [12,13] provide fine-gained isolation. They can provide a secure environment for applications even in a malicious OS. The function is implemented in the hypervisor, which intercepts the communication between applications and OS. In SplitVisor, it can be implemented in the GuestVisor.

Monitoring and Virus Detection. It is much more secure to implement monitoring and virus detection in the hypervisor. When an OS is under the attackers' control, all traditional virus detections are useless. The virus detection must be reliable, because it can access VMs' memory. But virus detection software are usually large, it is not suitable to be added to the TCB. In SplitVisor, it can be implemented in the GuestVisor. It is not possible to be added in other architecture without increasing the size of TCB. The monitoring software is the same as virus detection software.

Virtual TPM. A GuestVisor can emulate some devices for VMs, such as a virtual TPM. The TPM is an important device for software protection.

Table 6 summarizes the functions of the GuestVisors. All the functions are implemented in the hypervisor of Xen, as a part of TCB. It is more secure that SplitVisor can implement them out of the TCB. Other functions can be implemented in the GuestVisor as long as they do not violate the isolation restriction.

Table 6. Comparison of different designs - Where The Functions Are Implemented

Functions	SplitVisor	CloudVisor	Xen
I/O encryption	GuestVisor	CloudVisor	Hypervisor
VM life cycle	GuestVisor	CloudVisor	Hypervisor
Privileged instructions	GuestVisor	CloudVisor, hypervisor	Hypervisor
Fine-gained Isolation	GuestVisor	None	Hypervisor
Monitor and virus detection	GuestVisor	None	Hypervisor
Virtual TPM	GuestVisor	None	Hypervisor

4 Performance Evaluation

4.1 Memory Overhead

The memory overhead of SplitVisor is mainly caused by the GuestVisor. In general, users only need a small part of all functions. The GuestVisor is in users' memory space. If a users wants to save memory space, the user can choose a simple GuestVisor with less functions.

Xen occupies 64 MB memory. The GuestVisor has similar size. The memory of a typical VM is shown in Table 7, which is from Amazon EC2 [24]. The memory overhead is from 0.4% - 3.8%.

Table 7. Memory Overhead

VM type	VM	GuestVisor	Overhead
Small Instance	1.7 GB	64 MB	3.8%
Large Instance	7.5 GB	64 MB	0.9%
Extra Large Instance	15 GB	64 MB	0.4%

4.2 CPU and I/O Overhead

When running a VM, some privileged instructions and interrupts will cause VM exits, which introduces the major CPU and I/O overhead of SplitVisor when comparing with other approaches. In current hypervisor design, VM exits are delivered to and handled by the hypervisor. In SplitVisor, VM exits are transferred twice, from the SplitVisor to a GuestVisor. We firstly compare SplitVisor with other two-layer-hypervisor architectures: CloudVisor and nested hypervisor [17]. The upper part of Fig. 3 shows the process of VM exits in SplitVisor and CloudVisor. The labels in the figure indicate what messages are delivered. In CloudVisor, the hypervisor cannot access the memory of a VM. CloudVisor encrypts all the data from a VM to the hypervisor. In SplitVisor, all the memory of a VM are mapped in a GuestVisor, so the GuestVisor can easily deal with the VM's memory. The nested hypervisor provides full virtual environment for every

level, where the L1 hypervisor does not know that it is in a virtual machine. So the L1 hypervisor may execute some privileged instructions which traps to the L0 hypervisor. The L0 hypervisor emulates hardware for L1 the hypervisor. The SplitVisor does not aim at providing a virtual environment for GuestVisors. A GuestVisor is aware of virtualization. The lower part of Fig. 3 shows the process of VM exits in SplitVisor and nested hypervisor. When a VM exit occurs in a VM (L2), it is delivered to a GuestVisor (L1). A GuestVisor usually handles the VM exit by itself, while an L1 hypervisor may cause many new VM exits.

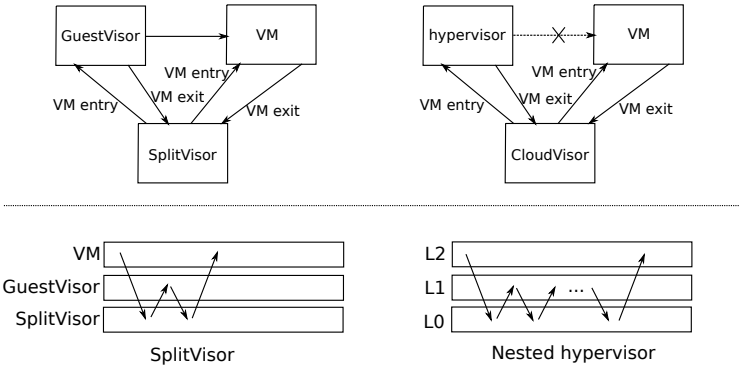

Fig. 3. Process of VM exit

A GuestVisor can control what can be intercepted by modifying a VM's VMCS. In our experiments, we assume that a GuestVisor intercepts the same VM exits as Xen. Then SplitVisor needs twice as many VM exits as Xen does. We choose SPECjbb2005 [32] as the benchmark program. We run SPECjbb2005 in Xen HVM with different numbers of JVMs. Then we count the number of VM exits and estimate the running time in SplitVisor. The results are shown in Fig 4. The average overhead is about 4.3%.

5 Related Work

Research similar to ours can be classified into several categories.

Hardware Level Protection: Some work, such as XOM [19,20], AEGIS [21] and AISE [22], can defend against hardware attacks. They use a specially designed CPU to defend against memory tampering. In these architecture, CPU encrypts all data that goes out of the CPU, and decrypts what are loaded into CPU. CPU maintains a hash tree that ensures data integrity. These work can protect all software from hardware based attacks, but do not address security problems inside the software.

Application Protection: Many vulnerabilities have been discovered in OSs. Many techniques, such as Overshadow [12,33] and SecureME [13], protect applications from a malicious OS. These work leverage the hypervisor to prevent

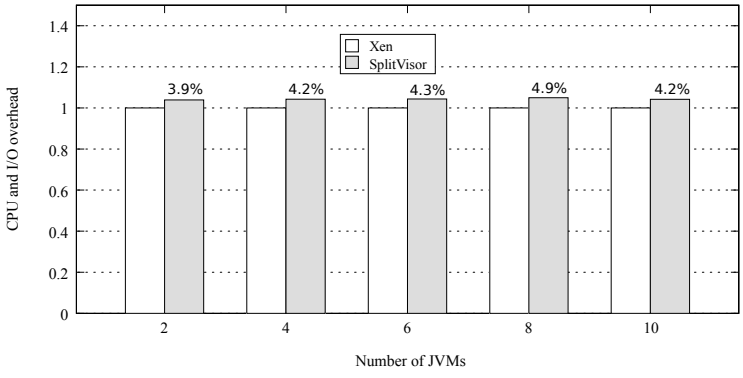

Fig. 4. CPU and I/O Overhead

the OS from directly accessing applications' memory and intercept the communication between the OS and applications. Bastion [15] provides module-level protection. Similar to application-level protection, entering into a module and going out of a module are intercepted by the hypervisor.

Hypervisor Protection: Hypervisor is the most important part in a virtualization architecture. TPM and other hardware protections [19,20,21,22] can verify the integrity of software when a hypervisor is loaded. HyperSentry [34] can verify the integrity dynamically. HyperSafe [35] can verify the integrity of control-flow of hypervisor execution. These work mainly focus on detecting attacks, instead of building a secure hypervisor. NoHype [4,5] cuts off the communication between hypervisor and VM after VM is booted. However, it cannot defend against the attacks from the management VM. So NoHype still has a large TCB.

VM Protection: The best way to protect a VM is to protect the hypervisor. If a hypervisor works correctly, the VM can be well protected by the hypervisor. Some work can provide protection without the help of a hypervisor. SICE [14] implements the protection mechanism in the SMM. Even if a hypervisor is malicious, a VM can be protected by SICE. Currently, SICE has a limit of protecting at most one VM on each core.

The turtles project [17] shows the architecture of nested hypervisors, which allows running hypervisors above a hypervisor. CloudVisor [16] leverages this architecture to protect VMs. In these designs, the orignial hypervisor is not at the highest level. All communications between a hypervisor and VMs must be verified by the CloudVisor, and the hypervisor cannot directly access VMs' memory. CloudVisor does not provide the protection of the hypervisor, because all requests from VM are forwarded to the hypervisor.

6 Conclusion

Existing virtualization architectures usually have either rich functions or a small TCB, but not both. In this paper, we propose an innovative virtualization

architecture, SplitVisor, to support both. SplitVisor has a two-layer-virtualization structure: a SplitVisor and GuestVisors. The SplitVisor is responsible for isolation between different users' VMs. A GuestVisor is responsible for emulating hardware for VMs. A GuestVisor is not required to reside in the TCB, so the TCB of SplitVisor is small. SplitVisor allows users to choose their own hypervisors. It cannot be achieved in single hypervisor designs where all VMs share the same hypervisor. The TCB of SplitVisor is stable because we do not add new functions to the SplitVisor, but to the GuestVisors. This also offers the opportunity to store the TCB in firmware or to optimize it with hardware.

Acknowledgement. This work was supported in part by NSF Grants CNS-1100221 and CNS-0905153.

References

1. "Xen hypervisor project", `http://www.xen.org/products/xenhyp.html`
2. Neiger, G., Santoni, A., Leung, F., Rodgers, D., Uhlig, R.: Intel virtualization technology: Hardware support for efficient processor virtualization. Intel Technology Journal 10(3), 167–177 (2006)
3. AMD. Secure virtual machine architecture reference manual
4. Keller, E., Szefer, J., Rexford, J., Lee, R.: Nohype: virtualized cloud infrastructure without the virtualization. In: Proceedings of the 37th Annual International Symposium on Computer Architecture, pp. 350–361. ACM (2010)
5. Szefer, J., Keller, E., Lee, R., Rexford, J.: Eliminating the hypervisor attack surface for a more secure cloud. In: Proceedings of the 18th ACM Conference on Computer and Communications Security, pp. 401–412. ACM (2011)
6. Kortchinsky, K.: Hacking 3d (and breaking out of vmware). BlackHat USA (2009)
7. Cve-2007-4993: Xen guest root can escape to domain 0 through pygrub (2007), `http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-4993`
8. Cve-2007-5497: Vulnerability in xenserver could result in privilege escalation and arbitrary code executionr (2007), `http://support.citrix.com/article/CTX118766`
9. Wojtczuk, R.: Subverting the xen hypervisor. BlackHat USA (2008)
10. Cve-2008-2100: Vmware buffer overflows in vix api let local users execute arbitrary code in host os (2008), `http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-2100`
11. Ristenpart, T., Tromer, E., Shacham, H., Savage, S.: Hey, you, get off of my cloud! exploring information leakage in third-party compute clouds. Computer and Communications Security (2009)
12. Chen, X., Garfinkel, T., Lewis, E., Subrahmanyam, P., Waldspurger, C., Boneh, D., Dwoskin, J., Ports, D.: Overshadow: a virtualization-based approach to retrofitting protection in commodity operating systems. In: ACM SIGARCH Computer Architecture News, vol. 36, pp. 2–13. ACM (2008)
13. Chhabra, S., Rogers, B., Solihin, Y., Prvulovic, X., Chen, M., Garfinkel, T., Lewis, E., Subrahmanyam, P., Waldspurger, C., Boneh, D., Dwoskin, J., Ports, D.: Secureme: a hardware-software approach to full system security. In: Proceedings of the International Conference on Supercomputing, pp. 108–119. ACM (2011)

14. Zhang, X., Azab, A., Ning, P.: Sice: A hardware-level strongly isolated computing environment for x86 multi-core platforms. In: 18th ACM Conference on Computer and Communications Security (2011)
15. Champagne, D., Lee, R.: Scalable architectural support for trusted software. In: 2010 IEEE 16th International Symposium on High Performance Computer Architecture (HPCA), pp. 1–12. IEEE (2010)
16. Zhang, F., Chen, J., Chen, H., Zang, B.: Cloudvisor: retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization. In: Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles, pp. 203–216. ACM (2011)
17. Ben-Yehuda, M., Day, M., Dubitzky, Z., Factor, M., Har'El, N., Gordon, A., Liguori, A., Wasserman, O., Yassour, B.: The turtles project: Design and implementation of nested virtualization. In: 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI), Vancouver, British Columbia, Canada, pp. 423–436 (October 2010)
18. Goldberg, R.: Architecture of virtual machines. In: Proceedings of the Workshop on Virtual Computer Systems, pp. 74–112. ACM (1973)
19. Lie, D., Thekkath, C., Mitchell, M., Lincoln, P., Boneh, D., Mitchell, J., Horowitz, M.: Architectural support for copy and tamper resistant software. ACM SIGPLAN Notices 35(11), 168–177 (2000)
20. Lie, D., Thekkath, C., Horowitz, M.: Implementing an untrusted operating system on trusted hardware. ACM SIGOPS Operating Systems Review 37(5), 178–192 (2003)
21. Suh, G., Clarke, D., Gassend, B., Van Dijk, M., Devadas, S.: Aegis: architecture for tamper-evident and tamper-resistant processing. In: Proceedings of the 17th Annual International Conference on Supercomputing, pp. 160–171. ACM (2003)
22. Chhabra, S., Rogers, B., Solihin, Y., Prvulovic, M.: Making secure processors os- and performance-friendly. ACM Transactions on Architecture and Code Optimization (TACO) 5(4), 16 (2009)
23. Huang, A.: Hacking the Xbox: an introduction to reverse engineering. No Starch Pr. (2003)
24. Amazon elastic compute cloud, `http://aws.amazon.com/`
25. Eucalyptus cloud computing software, `http://www.eucalyptus.com/`
26. Flexiscale cloud computing services, `http://www.flexiscale.com/`
27. Nimbus platform, `http://www.nimbusproject.org/`
28. Rackspace hosting, `http://www.rackspace.com/`
29. Xen users' manual v3.3, `http://www.xen.org/products/xenhyp.html`
30. Witteman, M., Oostdijk, M.: Secure application programming in the presence of side channel attacks. In: RSA Conference, vol. 2008 (2008)
31. Tpm main specification, `http://www.trustedcomputinggroup.org/`
32. Specjbb2005 (java server benchmark), `http://www.spec.org/jbb2005/`
33. Yang, J., Shin, K.: Using hypervisor to provide data secrecy for user applications on a per-page basis. In: Proceedings of the Fourth ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, pp. 71–80. ACM (2008)
34. Azab, A., Ning, P., Wang, Z., Jiang, X., Zhang, X., Skalsky, N.: Hypersentry: Enabling stealthy in-context measurement of hypervisor integrity. In: Proceedings of the 17th ACM Conference on Computer and Communications Security, pp. 38–49. ACM (2010)
35. Wang, Z., Jiang, X.: Hypersafe: A lightweight approach to provide lifetime hypervisor control-flow integrity. In: 2010 IEEE Symposium on Security and Privacy, pp. 380–395. IEEE (2010)