Secure Out-of-band Remote Management of Virtual Machines with Transparent Passthrough

Shota Futagami
Kyushu Institute of Technology
shota@ksl.ci.kyutech.ac.jp

Tomoya Unoki
Kyushu Institute of Technology
unoki@ksl.ci.kyutech.ac.jp

Kenichi Kourai
Kyushu Institute of Technology
kourai@ksl.ci.kyutech.ac.jp

ABSTRACT

Infrastructure-as-a-Service clouds provide out-of-band remote management for users to access their virtual machines (VMs). Out-of-band remote management is a method for indirectly accessing VMs via their virtual devices. While virtual devices running in the virtualized system are managed by cloud operators, not all cloud operators are always trusted in clouds. To prevent information leakage from virtual devices and tampering with their I/O data, several systems have been proposed by trusting the hypervisor in the virtualized system. However, they have various issues on security and management. This paper proposes *VSBypass*, which enables secure out-of-band remote management outside the virtualized system using a technique called *transparent passthrough*. VSBypass runs the entire virtualized system in an outer VM using *nested virtualization*. Then it intercepts I/O requests of out-of-band remote management and processes those requests in *shadow devices*, which run outside the virtualized system. We have implemented VSBypass in Xen for the virtual serial console and GUI remote access. We confirmed that information leakage was prevented and that the performance was comparable to that in traditional out-of-band remote management.

KEYWORDS

Virtual machines, Virtualized systems, Remote management, Information leakage, Nested virtualization

ACM Reference Format:
Shota Futagami, Tomoya Unoki, and Kenichi Kourai. 2018. Secure Out-of-band Remote Management of Virtual Machines with Transparent Passthrough. In *2018 Annual Computer Security Applications Conference (ACSAC '18), December 3–7, 2018, San Juan, PR, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3274694.3274749

1 INTRODUCTION

In Infrastructure-as-a-Service (IaaS) clouds, users can use a necessary number of virtual machines (VMs) to construct large systems. To manage their VMs provided by clouds, users access their VMs from remote hosts using remote management software such as SSH and VNC. In addition to such a management method for directly accessing VMs, clouds provide a method called *out-of-band remote management*. This management method enables users to indirectly access VMs via their virtual devices such as virtual serial devices, keyboards, and video cards in the virtualized system. The advantage of this method is that users can manage the systems in VMs even on their network configuration errors and at boot time. This is because out-of-band remote management does not rely on VMs' network or remote management servers running in VMs.

On the other hand, virtual devices in the virtualized system are managed by cloud operators, who may be untrusted in clouds [5, 12, 14, 17, 22, 31]. If there are malicious cloud operators, they can easily eavesdrop on and tamper with I/O for out-of-band remote management. To prevent information leakage from virtual devices and tampering with their I/O data, several systems have been proposed by trusting the hypervisor inside the virtualized system [5, 7, 13]. However, it is relatively easy for cloud operators resident in the same virtualized system to attack the hypervisor. To trust the hypervisor, cloud operators cannot manage the entire virtualized system after all. In addition, their applicability is not high because it is required that the hypervisor is clearly separated from the other components in the virtualized system.

This paper proposes *VSBypass* for enabling secure out-of-band remote management outside the virtualized system, using a technique called *transparent passthrough*. VSBypass uses *nested virtualization* [4] and runs the entire virtualized system in an outer VM. When a user VM issues I/O requests to virtual devices, VSBypass intercepts them and securely processes them in *shadow devices* outside the virtualized system. Since I/O for out-of-band remote management is completely processed outside the virtualized system, untrusted cloud operators inside the virtualized system cannot eavesdrop on or tamper with the I/O data. VSBypass does not need to trust the hypervisor inside the virtualized system. This means that cloud operators can manage the entire virtualized system. In addition, existing virtualized systems can be used because VSBypass does not basically depend on them.

We have implemented VSBypass in Xen 4.8.0 [2]. In the current implementation, VSBypass supports Xen and KVM [21] as a virtualized system running in an outer VM called the *cloud VM*. It can perform transparent passthrough of virtual serial devices, keyboards, mice, and video cards. When a user VM accesses these virtual devices, a VM exit occurs directly to the outside of the cloud VM. Shadow devices for processing the intercepted I/O are provided by a *proxy VM*, which is a thin VM created for each user VM. When virtual interrupts occur in shadow devices, they are efficiently redirected to the corresponding user VM using shared memory. We confirmed that VSBypass could prevent information leakage from virtual devices and that the performance was comparable to that in the traditional out-of-band remote management.

ACSAC '18, December 3–7, 2018, San Juan, PR, USA

ACM ISBN 978-1-4503-6569-7/18/12…$15.00
https://doi.org/10.1145/3274694.3274749

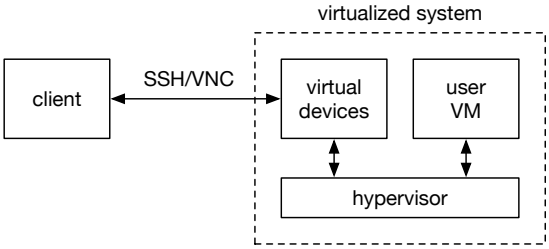


Figure 1: Out-of-band remote management of a VM.

The organization of this paper is as follows. Section 2 discusses information leakage and tampering in out-of-band remote management and four issues of previous approaches. Section 3 proposes VSBypass to resolve the issues and Section 4 describes the implementation. Section 5 compares VSBypass with the traditional remote management and shows the experimental results. Section 6 describes related work and Section 7 concludes this paper.

2 OUT-OF-BAND REMOTE MANAGEMENT

Out-of-band remote management is a method for indirectly accessing VMs via their virtual devices running in the virtualized system, as illustrated in Fig. 1. Users access virtual devices of VMs using remote management clients such as SSH and VNC and perform I/O for remote management. For out-of-band remote management, virtual serial devices, keyboards, mice, and video cards are used. When a remote client receives input data such as console and keyboard inputs from the user, it sends and writes the data to a virtual device. If a VM attempts to read that data from the virtual device, the hypervisor intercepts that access, obtains the data from the virtual device, and returns it to the VM. Similarly, if a VM attempts to write output data such as console and video outputs to a virtual device, the hypervisor intercepts that access and writes the data to the virtual device. Then, the data is read from the virtual device and is sent to a remote client.

Virtual devices used for out-of-band remote management are managed by cloud operators. Cloud operators are system administrators who are responsible for managing the entire virtualized systems in clouds. The virtualized system consists of the hypervisor, privileged components including virtual devices, and user VMs. Unfortunately, not all the cloud operators are always trusted. It is reported that 28% of cybercrimes are caused by insiders [20]. One example of insiders is malicious system administrators, who attack systems actively. In 2010, a site reliability engineer in Google violated user's privacy [27]. Another example is curious but honest system administrators, who may eavesdrop on attractive information that they can easily obtain from user VMs. It is revealed that 35% of system administrators have accessed sensitive information without authorization [6].

Untrusted cloud operators can easily eavesdrop on and tamper with I/O data for out-of-band remote management via virtual devices of user VMs. Consequently, for example, login passwords typed by users may leak and, after login with the stolen password, illegal commands may be executed in the user VM. To hide the execution of such illegal commands, the video screen of the user VM can be replaced with a normal one. To prevent such attacks, cloud operators should be granted only the least privilege. However, it is often difficult to do that because legitimate operations can be also used for malicious purposes. If cloud operators can manage virtual devices, they can also compromise them.

To prevent such attacks, several systems trusting the hypervisor have been proposed. For example, FBCrypt [7] and SCCrypt [13] encrypt input data in VNC and SSH clients, respectively. When a user VM attempts to obtain the data from a virtual device, the hypervisor decrypts it transparently. When a user VM attempts to write output data to a virtual device, the hypervisor encrypts it and the remote client decrypts it. FBCrypt can also detect tampering with input data using message authentication code. As such, cloud operators cannot eavesdrop on or tamper with the I/O data because virtual devices process only encrypted data. Note that using the trusted hypervisor can also prevent the leakage of such I/O data from user VMs by encrypting their memory [14, 15, 25]. Although user VMs have to handle decrypted input data and unencrypted output data, cloud operators cannot eavesdrop on such data inside user VMs even by using VM introspection [8] outside the VMs.

Unlike these systems, SSC [5] securely encrypts I/O data in users' special VMs instead of the hypervisor. The VMs are called service domains (SDs) and are protected by the trusted hypervisor. It should be noted that the targets are not virtual devices used for out-of-band remote management but virtual disks. For example, encrypted data is read from a physical device by a virtual device and is decrypted by an SD. Data written to a physical device is first encrypted by an SD. Since SSC isolates user VMs from cloud operators by using the trusted hypervisor, all the cloud operators cannot access I/O data inside user VMs.

As such, the trusted computing base (TCB) of these systems includes the hypervisor at least. In other words, it is assumed that untrusted cloud operators in the virtualized system can compromise privileged components including virtual devices, except for the hypervisor. In addition, these systems depend on cryptography to prevent information leakage from virtual devices and tampering with their I/O data. Therefore, the following four issues arise.

Attacks against the hypervisor inside the virtualized system. To enable privileged components to run on top of the hypervisor, the hypervisor provides rich management interfaces. This means that the hypervisor is tightly coupled with privileged components, as illustrated in Fig. 2. If untrusted cloud operators abuse such interfaces, the trusted hypervisor can be easily compromised if it has vulnerabilities [19, 23]. Consequently, encrypted I/O data can be decrypted by using cryptographic keys stored in the hypervisor. Illegal data can be also injected into out-of-band remote management.

Management of the virtualized system. It has to be guaranteed that untrusted cloud operators cannot manage the trusted hypervisor. If the hypervisor could be freely replaced with malicious one, it would be no longer trusted. However, it is unrealistic to manage the hypervisor and the rest of the virtualized system independently. Since the hypervisor is tightly coupled with privileged components as described above, the virtualized system has to be updated as a whole. If only privileged components were updated, the integrity of the virtualized system would not be maintained.

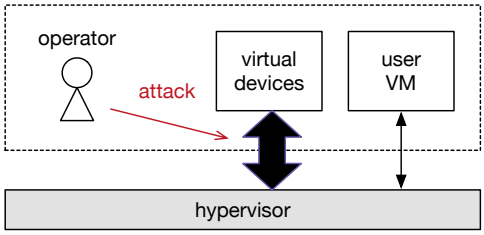


Figure 2: The attack surface in the system using the trusted hypervisor.

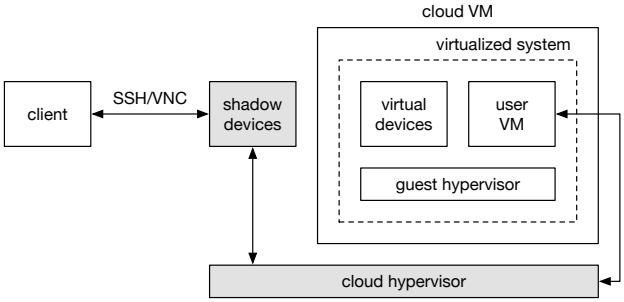


Figure 3: The system architecture of VSBypass. Shaded components are the TCB.

This means that cloud operators cannot maintain the virtualized system at all as a result.

Feasibility of trusting the hypervisor. To trust only the hypervisor, it is necessary that the hypervisor is clearly separated from the other privileged components. Examples of such virtualized systems are Xen, vSphere, and Hyper-V, which use bare-metal hypervisors running directly on top of hardware. In contrast, since KVM runs the hypervisor inside the host operating system, it is difficult to separate only the hypervisor. Consequently, the entire host operating system has to be trusted. However, the operating system is much more complex than the hypervisor and therefore the TCB becomes much larger.

Cryptography management. Cryptographic keys are necessary for encryption, but it is not easy to securely manage keys in practice. If the key management system has vulnerabilities, encryption would be useless. In addition, remote management clients need to be modified to provide an encryption mechanism of input data and a decryption mechanism of output data. To preserve the integrity of I/O data, they need a mechanism for calculating the message authentication code of I/O data. Since new clients have to be developed to support these mechanisms, users cannot use the existing clients including commercial products.

3 VSBYPASS

3.1 Assumptions and Threat Model

We assume that cloud operators inside the virtualized system may be untrusted. It is difficult to trust all of the cloud operators in large-scale IaaS clouds. Unlike previous work [5, 7, 13], cloud operators have full control over the entire virtualized system, including not only virtual devices but also the hypervisor. They can compromise any components inside the virtualized system and eavesdrop on and tamper with I/O data of out-of-band remote management for user VMs.

In contrast, we assume that cloud providers are trusted. Since a bad reputation is critical for cloud providers, this assumption is widely accepted [5, 12, 14, 17, 22, 31]. Cloud providers maintain hardware and restrict physical access by untrusted cloud operators. They also maintain the TCB for VSBypass, whose details are described in the next subsection. For these purposes, cloud providers should have a few trusted system administrators. We assume that such administrators do not perform malicious activities because they are proud of their jobs and are rewarded adequately. A few trusted administrators and many untrusted operators form an administrative hierarchy, which is also adopted in many systems, e.g., SYSDBA and SYSOPER in Oracle Database [18] and eight types of administrators in IBM Domino [9].

In this paper, we do not consider direct attacks against user VMs to eavesdrop on or tamper with I/O data of out-of-band remote management. Such attacks can be prevented by previous work [31], which is discussed in the next subsection. Denial-of-service attacks against out-of-band remote management is out of the scope of this paper because they do not cause information leakage or tampering.

3.2 Architecture

VSBypass achieves secure out-of-band remote management outside the virtualized system using a mechanism called *transparent passthrough*. VSBypass runs the entire virtualized system in an outer VM using *nested virtualization* [4]. It intercepts I/O requests of user VMs and securely processes them in *shadow devices* running outside the virtualized system. This enables users to perform out-of-band remote management without relying on the virtualized system. Therefore, I/O data of out-of-band remote management does not leak to untrusted cloud operators inside the virtualized system. Also, cloud operators cannot tamper with the I/O data in the virtualized system.

Fig. 3 illustrates the system architecture of VSBypass. The virtualized system including user VMs runs in an outer VM called the *cloud VM*. Shadow devices run outside the cloud VM and process I/O requests of user VMs like virtual devices inside the virtualized system. Users can access shadow devices, instead of virtual devices, for secure out-of-band remote management. The cloud VM and shadow devices run on top of the hypervisor called the *cloud hypervisor*. To distinguish two hypervisors, we call the original hypervisor inside the virtualized system the *guest hypervisor*.

The TCB for VSBypass is the cloud hypervisor and shadow devices. These components are remotely attested with TPM by remote users or the trusted third party. The TCB and the cloud VM are maintained only by trusted system administrators in cloud providers and cannot be accessed by any cloud operators. Since the entire virtualized system is isolated from the TCB by the cloud

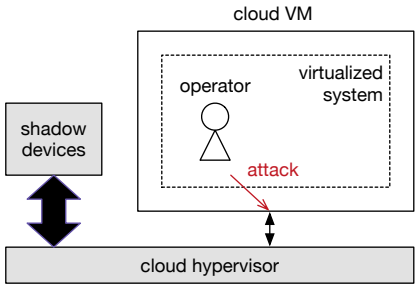


Figure 4: The attack surface in VSBypass.

VM more strictly in VSBypass, it is easier to grant the least privilege to cloud operators. To prevent attacks against user VMs, we can use CloudVisor [31], which also uses nested virtualization. In CloudVisor, the cloud hypervisor called the security monitor restricts access to user VMs by cloud operators. In this paper, we focus on attacks against virtual devices and the guest hypervisor in the virtualized system, the integration of VSBypass and CloudVisor is our future work.

In VSBypass, out-of-band remote management is performed as follows. For input, a remote client sends and writes input data to a shadow device via SSH or VNC. When the user VM attempts to access a virtual device to obtain that input data, the cloud hypervisor directly intercepts that access without intervention by the guest hypervisor. It obtains input data from the corresponding shadow device and returns that data to the user VM. For output, when a user VM attempts to write output data to a virtual device, the cloud hypervisor directly intercepts that access. Then it passes that data to the corresponding shadow device. A remote client receives the output data from the shadow device via SSH or VNC.

VSBypass can resolve the four issues in out-of-band remote management, which are described in Section 2.

Smaller attack surface to the TCB. It is more difficult for untrusted cloud operators to attack the TCB, i.e., the cloud hypervisor and shadow devices outside the cloud VM. This is because the entire virtualized system is confined in the cloud VM, as illustrated in Fig. 4. The attack surface from the cloud VM to the cloud hypervisor is much smaller than that from privileged components to the guest hypervisor. The cloud hypervisor provides only the hardware interface to the cloud VM, which is the boundary of virtualization.

Management of the entire virtualized system. Cloud operators can manage the entire virtualized system including the guest hypervisor because VSBypass does not need to trust the guest hypervisor inside the virtualized system. They can update not only privileged components but also the guest hypervisor although the validity of their management has to be checked by trusted cloud providers. Consequently, VSBypass enables cloud providers to clearly separate the responsibility of system management at the boundary of virtualization.

Support for various virtualized systems. Any virtualized systems are available in VSBypass because the entire virtualized system is virtualized using the cloud VM. Specifically, VSBypass can easily support not only bare-metal hypervisors but also hosted hypervisors. To perform out-of-band remote management, VSBypass does basically not depend on the internals of the virtualized system.

No cryptography. Cryptographic key management is not necessary because VSBypass does not rely on encryption to prevent information leakage and tampering. VSBypass does not need to encrypt I/O data of out-of-band remote management or calculate message authentication code of them. Therefore, users can use the existing remote management software.

In terms of performance, the approach of using nested virtualization is feasible because it is reported that the overhead is 6–8% for common workloads [4]. Special-purpose cloud hypervisors as used in CloudVisor [31] and TinyChecker [26] can improve the performance of nested virtualization more. Recently, hardware support for nested virtualization has been also added. For example, Intel VMCS Shadowing [10] can eliminate VM exits due to VMREAD and VMWRITE instructions for accessing virtual machine control structure (VMCS). The ARMv8.3 architecture [1] supports nested virtualization and its extension called NEVE has been proposed for coalescing and deferring traps [16].

4 IMPLEMENTATION

We have implemented VSBypass in Xen 4.8.0 [2]. We run the hypervisor where VSBypass is implemented as the cloud hypervisor and the unmodified hypervisor as the guest hypervisor in the cloud VM. VSBypass currently supports Xen and KVM as virtualized systems. We run shadow devices as QEMU [3] in the management VM called Dom0. VSBypass supports virtual serial devices, keyboards, mice, and video cards as shadow devices.

4.1 Proxy VM

VSBypass runs a VM called a *proxy VM* per user VM on top of the cloud hypervisor, as illustrated in Fig. 5. A proxy VM is a VM used only for providing virtual devices as shadow devices to a user VM. Users can transparently perform secure out-of-band remote management of a user VM by specifying the ID of the corresponding proxy VM and accessing its virtual devices. VSBypass assigns the same number of virtual CPUs (vCPUs) as the cloud VM to each proxy VM. The vCPUs of a proxy VM are mapped to those of the cloud VM one-to-one to easily redirect I/O requests, as described in Section 4.2. In contrast, VSBypass assigns minimum amounts of memory and disk used only for the boot of a proxy VM and does not assign NICs.

Using proxy VMs is not so costly for IaaS providers. After a proxy VM is booted, it is paused except for virtual devices used as shadow devices. Since any vCPUs are not scheduled, the proxy VM does not consume CPU time at all after the boot. Similarly, the bandwidth of memory and disk is not consumed at all. The virtual devices of the proxy VM consume CPU time, memory, and the network bandwidth, whereas those of the corresponding user VM are not used instead because they are bypassed by VSBypass. The management cost of these resources increases a bit, but that can be justified by the increase in security.

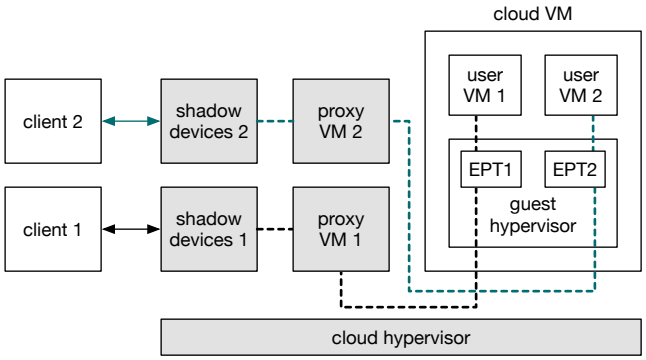


Figure 5: Shadow devices provided by a proxy VM for each user VM. Shaded components are added to the traditional virtualized system.

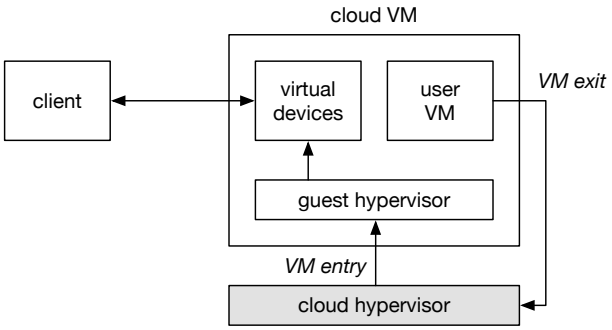


Figure 6: Traditional I/O processing in nested virtualization.

To map a user VM to a proxy VM in the cloud hypervisor, VSBypass identifies a user VM using the extended page tables (EPT), which are created for each user VM. This is because a user VM is an abstraction inside the virtualized system and the cloud hypervisor cannot directly identify a user VM. VSBypass obtains the EPT address from the VMCS of a user VM when it intercepts I/O requests in the user VM. If the address is a newly detected one, VSBypass associates it with a new proxy VM. In the current implementation, VSBypass creates several proxy VMs in advance and assigns one of them when detecting a new EPT address.

A user can securely specify his user VM using a *VM tag* [17] and obtain the ID of the corresponding proxy VM from the cloud hypervisor. A user VM registers a unique VM tag to the cloud hypervisor in advance using an *ultracall* [17]. An ultracall is a mechanism for directly invoking the cloud hypervisor from a user VM inside the virtualized system. It is achieved by executing the `vmcall` instruction to cause a VM exit to the cloud hypervisor. Then the cloud hypervisor associates the VM tag with the EPT address of the user VM.

4.2 I/O Interception

Using nested virtualization, I/O requests of a user VM are usually processed as illustrated in Fig. 6. When a user VM executes an I/O instruction, a VM exit occurs from the cloud VM running the user VM to the cloud hypervisor, not to the guest hypervisor. The cloud hypervisor performs a VM entry into the cloud VM to emulate that instruction in the guest hypervisor of the cloud VM. The guest hypervisor communicates with a virtual device in the cloud VM to process I/O. After that, when the guest hypervisor attempts to perform a VM entry into the user VM, a VM exit occurs to the cloud hypervisor. Finally, the cloud hypervisor performs a VM entry into the user VM.

In contrast, VSBypass does not forward I/O requests of a user VM to the guest hypervisor but completely process them outside the cloud VM, as illustrated in Fig. 3. When a user VM executes an I/O instruction, a VM exit occurs to the cloud hypervisor as usual. If that I/O request is for port-mapped I/O, the cloud hypervisor examines the port address used in the I/O instruction that has caused a VM exit. If the address is used by virtual serial devices, keyboards, mice, or video cards, the cloud hypervisor emulates that instruction. Table 1 lists such intercepted port addresses. If that I/O request is for memory-mapped I/O, the cloud hypervisor examines the memory address that has caused a VM exit by EPT violation. If the address is used by virtual video cards, the cloud hypervisor emulates that instruction. Such intercepted memory addresses are `A0000-BFFFF`, which is used for accessing video memory (VRAM). Otherwise, the cloud hypervisor performs a VM entry into the cloud VM as usual.

Table 1: Intercepted port addresses.

device	port addresses
serial device	`3F8-3FF`
keyboard/mouse	`60, 64`
video card	`1CE-1CF, 3B0-3BB, 3C0-3DF`

The cloud hypervisor redirects the intercepted I/O request to the corresponding shadow device, as in Fig. 7. It first obtains the EPT address from the VMCS of the user VM that has caused a VM exit and finds the proxy VM corresponding to the user VM. Then it identifies the proxy VM's vCPU corresponding to the cloud VM's vCPU used for I/O of the user VM. One of the vCPUs in the cloud VM is used to execute an I/O instruction in the user VM. Next, the cloud hypervisor copies the request in the I/O request buffer for the cloud VM's vCPU to the buffer for the proxy VM's. Finally, it blocks the cloud VM's vCPU and sends an event to the target shadow device.

VSBypass returns the result of the I/O processing to the user VM as follows. When the shadow device completes I/O processing, it attempts to send an event to the proxy VM's vCPU as usual. At this time, the cloud hypervisor intercepts that event and redirects it to the corresponding cloud VM's vCPU. First, it finds the cloud VM's vCPU and copies the result and state in the I/O request buffer for the proxy VM's vCPU to the buffer for the cloud VM's. Finally, it unblocks the cloud VM's vCPU and performs a VM entry into the user VM. As a result, the user VM's vCPU receives the result and continues to run from the next instruction of the intercepted I/O instruction.

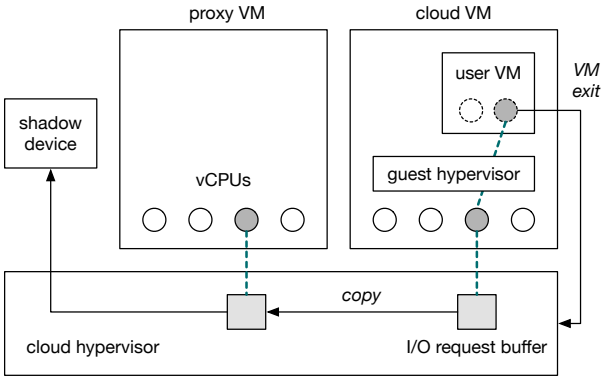


Figure 7: I/O processing in VSBypass.

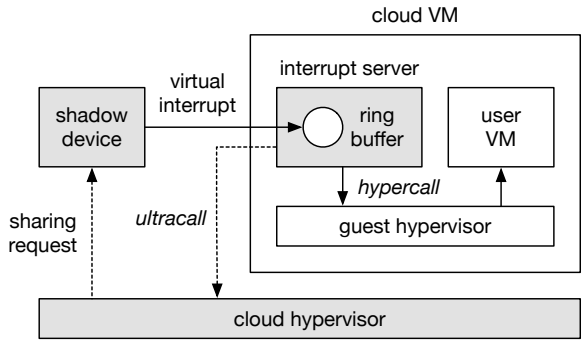


Figure 8: The delivery of virtual interrupts using shared memory.

4.3 Redirection of Virtual Interrupts

Virtual interrupts caused in shadow devices have to be redirected to user VMs. However, it is difficult to do that without relying on the guest hypervisor inside the virtualized system. If the interrupt mechanism is para-virtualized, only the guest hypervisor can send virtual interrupts to user VMs. Even if not, the virtual interrupt controller is often implemented in the guest hypervisor. To avoid relying on the virtualized system as much as possible, it is possible to communicate between a shadow device and an interrupt server running in the virtualized system and redirect virtual interrupts via the guest hypervisor. However, this method suffers from large overhead of the virtual network in nested virtualization. Although Xen-Blanket [30] reduces this overhead using the Blanket driver, it largely relies on the virtualized system because that driver has to be implemented in the operating system of Dom0 and a new hypercall has to be added to the guest hypervisor.

Therefore, VSBypass shares a ring buffer between a shadow device and the interrupt server using an ultracall, as illustrated in Fig. 8. Since an ultracall enables the interrupt server in the virtualized system to directly invoke the cloud hypervisor, it is unnecessary to modify the guest hypervisor for that. The cloud hypervisor sends information on the ring buffer passed by the ultracall to a shadow device and makes the interrupt server and the shadow device share the ring buffer. To prevent the memory page used for the ring buffer from being swapped out to a disk, the interrupt server pins allocated memory by issuing the mlock system call. For the redirection of virtual interrupts, the shadow device writes an IRQ and an assertion level of a virtual interrupt to the ring buffer. For example, the numbers of ISA IRQ are 4 for serial devices and 1 for keyboards.

To enable a shadow device to handle the memory address of the ring buffer passed from the interrupt server, the cloud hypervisor translates the virtual address of the ring buffer into a physical address. For this address translation, it obtains the value of the CR3 register from cloud VM's vCPU. Since the interrupt server runs in Dom0, whose vCPUs are para-virtualized, the register contains the address of the page directory for the interrupt server. Then the cloud hypervisor walks the page tables from the page directory and obtains the physical address of the ring buffer. Since the memory is also para-virtualized in Dom0, the obtained physical address is one of the cloud VM. The address is sent to the specified shadow device with a pseudo I/O request. The shadow device can map the memory page of the ring buffer by specifying the physical address to the hypercall.

To obtain information on virtual interrupts written by the shadow device, the interrupt server periodically polls the ring buffer. Using polling, it does not rely on the notification mechanism of virtual interrupts, which needs to modify the guest hypervisor and/or the operating system in Dom0. When the interrupt server detects the write of virtual interrupts, it reads that information and sends a virtual interrupt to the user VM via the guest hypervisor by issuing a hypercall. Since the redirection of virtual interrupts depends on the interrupt server and the guest hypervisor in an untrusted virtualized system, virtual interrupts may not be correctly delivered to user VMs. However, confidentiality is maintained because virtual interrupts include no sensitive information.

When KVM [21] is used as a virtualized system, the interrupt server is embedded into QEMU running on top of the host operating system. This is because only QEMU can invoke the guest hypervisor for injecting virtual interrupts into user VMs. Note that it is not necessary to modify the host operating system and the guest hypervisor in the virtualized system.

4.4 Sharing VRAM

The VRAM for a virtual video card in Xen is allocated as part of the memory of a user VM. Its physical address is fixed to `F1000000` and the size is 16 MB. A user VM can access the VRAM through the memory region of `A0000-BFFFF` as traditional. This memory-mapped I/O causes a VM exit by EPT violation to the hypervisor. The hypervisor invokes the virtual video card and the device can handle the I/O. In VSBypass, a VM exit occurs directly to the cloud hypervisor, as described in Section 4.2 and the shadow video card can handle the I/O.

Unfortunately, the user VM not only performs such memory-mapped I/O but also accesses the VRAM directly. Since such direct access of the VRAM does not cause a VM exit by default, the shadow video card cannot handle the I/O. If the cloud hypervisor intercepts all accesses to the VRAM, that imposes too large overhead. Traditionally, a virtual video card obtains data from the

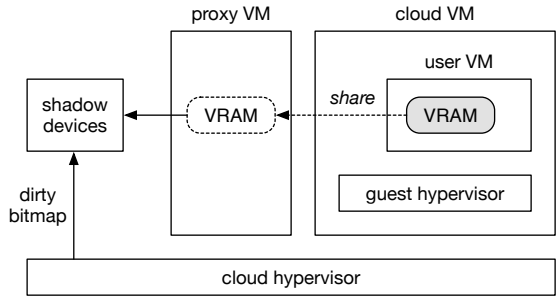


Figure 9: Sharing the VRAM of a user VM with a proxy VM.

VRAM if necessary, e.g., when it sends the screen data of a user VM to a VNC client. In addition, the shadow video card needs to access the VRAM after memory-mapped I/O is trapped. Therefore, a virtual video card has to be largely modified to develop the shadow one because it is more complicated to access the memory of a user VM from a shadow video card. Like the other shadow devices, it is desirable to reuse the code of existing virtual devices as much as possible.

For the ease of developing a shadow video card, a proxy VM shares VRAM with a user VM, as illustrated in Fig. 9. Using the shared VRAM, a shadow video card can obtain data in the VRAM of a user VM from the corresponding proxy VM as if the proxy VM were the user VM. To enable the VRAM of a user VM to be shared, the cloud hypervisor records a list of memory pages used for the VRAM at the first memory-mapped I/O by the user VM. At this time, it translates physical addresses of the memory pages in the user VM into those in the cloud VM using EPT so that the VRAM can be accessed by a shadow video card. After that, a shadow video card issues a new hypercall for obtaining the page list of the VRAM. Then it maps these pages and replaces the VRAM of the proxy VM with it. Note that untrusted cloud operators cannot access the VRAM by the memory protection of CloudVisor [31].

A shadow video card tracks updates of the shared VRAM by a user VM to detect updated screen areas. Traditionally, the hypervisor write-protects the VRAM pages of a user VM in EPT and records updated pages in a dirty bitmap. Then, a virtual video card periodically obtains the dirty bitmap from the hypervisor. In VSBypass, instead of the guest hypervisor, the cloud hypervisor write-protects the VRAM pages of a user VM in the EPT of the cloud VM and manages a dirty bitmap for the user VM. For this purpose, the cloud hypervisor provides a new hypercall to shadow video cards.

4.5 VM Migration

Using shadow devices makes it difficult to migrate a user VM. Since the migration manager in the virtualized system transfers the state of a user VM, it cannot obtain or transfer the state of shadow devices. If the state of shadow devices is lost at the destination host, remote users may not be able to connect to a migrated VM after VM migration. One possible solution is to migrate a user VM with the corresponding proxy VM by using a co-migration technique of VMs [11]. However, it is difficult to apply this technique because proxy VMs are out of control of the migration manager in virtualized system. It is risky to give a privilege for migrating proxy VMs to that migration manager.

Another possible solution is that the migration manager obtains the state of shadow devices running outside the virtualized system. To achieve this, it is necessary to develop a secure communication channel to shadow devices because this interface can become a new attack surface. This is our future work.

5 EXPERIMENTS

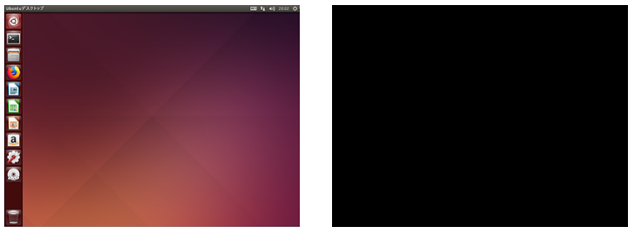
We conducted experiments to show the effectiveness of VSBypass. First, we attempted to eavesdrop on I/O for out-of-band remote management. Second, we measured the response time of input and the throughput of output in out-of-band remote management. For comparison, we used the traditional cloud without nested virtualization (Cloud) and the traditional virtual cloud with nested virtualization (vCloud).

We used a PC with an Intel Xeon E3-1290 v2 processor and 8 GB of memory. We assigned two or three vCPUs and 4 GB of memory to the cloud VM and two or three vCPUs and 1 GB of memory to a user VM in the cloud VM. The cloud VM ran Xen 4.4.0 as a virtualized system and the user VM and Dom0 ran Linux 3.13. As the cloud hypervisor, we ran modified Xen 4.8.0. To run a remote management client, we used a PC with an Intel Xeon E3-1270 processor and 8 GB of memory. These PCs were connected using Gigabit Ethernet. When using a virtual serial console, we used the OpenSSH 6.0p1 client. When using GUI remote access, we used TightVNC Java Viewer 2.0.95 [28].

5.1 Eavesdropping on I/O data

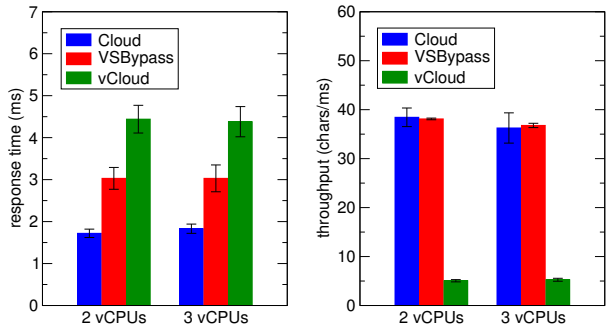
First, we attempted to eavesdrop on I/O data in a virtual serial device and the guest hypervisor of the virtualized system. To record a log in the device, we modified the device implemented in QEMU running inside the virtualized system. When the device processed access to I/O ports, it analyzed I/O access and recorded input and output characters in a log file. In addition, we modified the handler for processing VM exits caused by the execution of I/O instructions in the guest hypervisor. The modified handler analyzed I/O access and wrote input and output characters to a local console. Then, we connected to the SSH server and accessed a shadow or virtual serial device. In Cloud and vCloud, we could eavesdrop on all of the input and output characters. For VSBypass, in contrast, we could not obtain any I/O data.

Next, we attempted to eavesdrop on I/O data in virtual keyboard and video devices of the virtualized system. For keyboard inputs, we modified the keyboard device in QEMU and recorded input characters in a log file when the device processed input data. To record inputs in human-readable characters, we handled shift keys correctly and translated scancodes generated by a keyboard into X11 keysyms. For video outputs, we modified the video device in QEMU and recorded the contents of VRAM to a log file periodically. Then, we connected to the VNC server and accessed shadow or virtual keyboard and video devices. As a result, we could obtain all the input characters and capture the screen of the user VM in Cloud and vCloud, but we could not in VSBypass. The screens captured in three environments are shown in Fig. 10.



(a) Cloud **and** vCloud (b) **VSBypass**

Figure 10: The screens captured from virtual video devices.



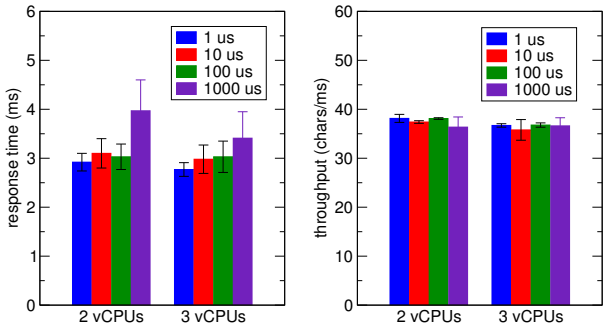
(a) Response time **(b) Throughput**

Figure 11: The performance of a virtual serial console.

5.2 Performance of a Virtual Serial Console

5.2.1 Response Time. We measured the response time of a virtual serial console using SSH. The response time was from when the SSH client sent a console input until it received a console output caused by its remote echo in the user VM. Fig. 11(a) shows the response time when we assigned the different number of vCPUs to the cloud VM. The response time in VSBypass was 1.3 ms longer than that in Cloud. This is because it took a longer time to process input data in the user VM due to the overhead of nested virtualization. The user VM had to process a received character and echo it back. In contrast, VSBypass was 1.4 ms shorter than vCloud. The virtual serial device inside the virtualized system suffered from the overhead of nested virtualization, while the shadow serial device outside it did not. The response time in VSBypass was not affected by the number of vCPUs although that in Cloud was 6% slower in three vCPUs.

5.2.2 Throughput. We measured the throughput of a virtual serial console using SSH. We printed the contents of a text file using the cat command in the user VM and measured the time from when the command was executed until the output was completed. Fig. 11(b) shows the throughput. Surprisingly, the throughput in VSBypass was almost the same as that in Cloud although the response time was slower. In this experiment, the shadow or virtual serial device processed a large amount of output data and dominated the throughput. Since the shadow serial device did not suffer from the overhead of nested virtualization in VSBypass, VSBypass could achieve high throughput. In contrast, the throughput in vCloud was 87% lower due to that overhead in the virtual serial



(a) Response time **(b) Throughput**

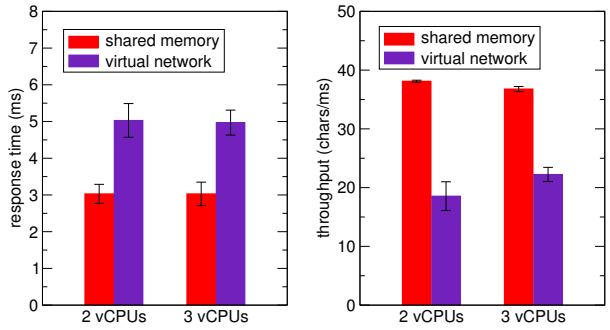
Figure 12: The impact of the polling interval.

device inside the virtualized system. When the number of vCPUs of the cloud VM increased, the throughput was degraded by 3.5% in VSBypass unexpectedly.

5.2.3 Impact of the Polling Interval. We examined the impact of the polling interval on the performance. At the specified polling interval, the redirection mechanism of virtual interrupts checked a ring buffer in shared memory. We measured the response time and throughput of a virtual serial console when we changed the polling interval between 1 and 1000 μs. As shown in Fig. 12, the response time was almost constant for the polling interval between 1 and 100 μs. However, it became clearly longer at the polling interval of 1000 μs, especially for the cloud VM with two vCPUs. This is because it took a longer time to deliver a virtual interrupt to the user VM after a console input was received by the shadow serial device. In contrast, the throughput was almost not affected by the polling interval. Therefore, we adopted the polling interval of 100 μs to reduce the polling overhead as much as possible.

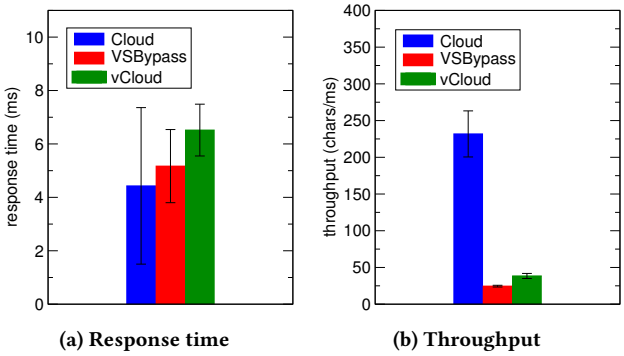
5.2.4 Performance of Interrupt Redirection. We compared the performance of the redirection mechanism of virtual interrupts. For this purpose, we have developed a variant of VSBypass, which redirected virtual interrupts using the virtual network, as described in Section 4.3, instead of using shared memory. Fig. 13 shows the response time and throughput of a virtual serial console using SSH when we used shared memory or the virtual network. Using the virtual network, the response time was only 2.0 ms longer due to the extra overhead. However, the throughput degradation was 40-51% because the virtual network device inside the virtualized system suffered from the overhead of nested virtualization. In this experiment, the number of redirected virtual interrupts was two for printing each character. Increasing the number of vCPUs slightly improved the throughput in VSBypass.

5.2.5 Using KVM as a Virtualized System. We examined the performance when we used KVM as a virtualized system. In this experiment, we used a PC with an Intel E3-1226 v3 processor and 8 GB of memory. We assigned two vCPUs and 3 GB of memory to the cloud VM and two vCPUs and 1 GB of memory to a user VM. The cloud VM ran Linux 4.2 and QEMU-KVM 2.4.1, and the user VM ran Linux 4.2.



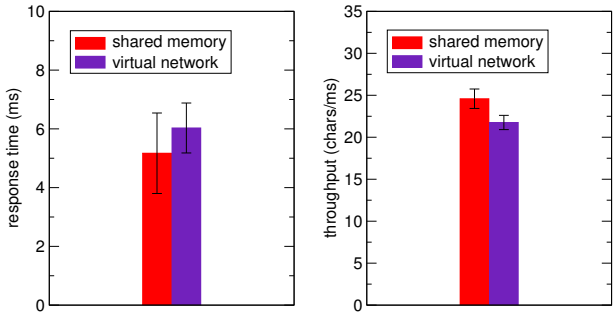
(a) Response time (b) Throughput

Figure 13: The comparison to network-based interrupts.



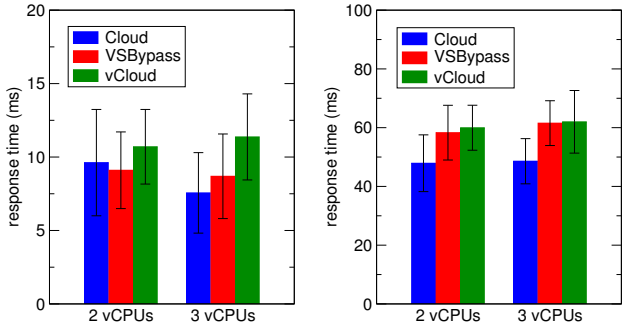
(a) Response time (b) Throughput

Figure 14: The performance of a virtual serial console for KVM.



(a) Response time (b) Throughput

Figure 15: The comparison to network-based interrupts for KVM.



(a) Text mode (b) Graphic mode

Figure 16: The response time of GUI remote access.

First, we measured the response time and the throughput. As shown in Fig.14(a), the trend of the response time was similar to that for Xen in Fig. 11(a). However, the performance difference was smaller, while the variance was larger. In contrast, the trend of the throughput was largely different, as shown in Fig. 14(b). The throughput in Cloud was much better, even compared with that for Xen in Fig. 11(b). This is probably because there is some difference in a virtual serial console between KVM and Xen. In addition, the throughput in VSBypass was lower than that in vCloud and was the worst. One of the reasons is that our redirection mechanism of virtual interrupts is unstable for KVM due to too frequent interrupts. Fixing this problem is our future work.

Next, we compared the performance of the redirection mechanism of virtual interrupts. As shown in Fig. 15, it is shown that the performance difference was relatively small. Using shared memory, the response time became 0.9 seconds shorter, but the throughput increased only by 13%. One possible reason is that the virtual network in KVM is faster than in Xen, but the biggest reason would be the implementation issue in our redirection mechanism for KVM.

5.3 Performance of GUI Remote Access

5.3.1 Response Time. We measured the response time of a keyboard input in out-of-band GUI remote access using VNC. When a user pressed a key, the VNC client first sent the keyboard input to the user VM via the VNC server and a shadow or virtual keyboard device. Then, the input was processed in the user VM and the input character was displayed in the text console or GUI terminal, depending on the screen mode. Finally, the VNC server sent the updated screen data in VRAM to the VNC client. The response time was the time from when the VNC client received a keyboard input until it received updated screen data from the server.

Fig. 16(a) shows the response time in the text and graphic modes. When we used the text mode, the response time in VSBypass was 1.1 ms longer than Cloud for the cloud VM with three vCPUs. This result is similar to that for a virtual serial console in Fig. 11(a). However, the response time was 0.5 ms shorter for the cloud VM with two vCPUs. This is probably due to the complex implementation of screen update using a timer in VNC. In contrast, when we used the graphic mode, the response time in VSBypass was clearly longer and was almost the same as that in vCloud. This is because the processing overhead in the user VM dominated the response time. When we increased the number of vCPUs, the response time decreased in the text mode but slightly increased in the graphic mode.

5.3.2 Screen Update Time. We measured the update time of the screen data. In VNC, after the VNC client sent a request for screen update, the VNC server returned updated screen data if the screen

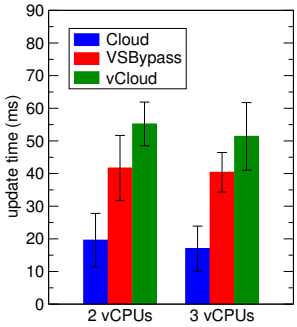


Figure 17: The screen update time of GUI remote access.

was updated. The update time was the time from when the VNC client sent the request until it received updated screen data. To always update the screen of the user VM, we drew a rectangle of 300×600 with random colors every second in the user VM. Fig. 17 shows the screen update time in the graphic mode. The update time in VSBypass was 23 ms longer than Cloud. Like in the above experiment, the processing in the user VM became a bottleneck in VSBypass. Increasing the number of vCPUs slightly improved the performance.

6 RELATED WORK

Device passthrough is a mechanism for directly accessing physical devices from VMs. This mechanism is used to reduce the overhead of device virtualization and improve I/O performance of VMs. Passthrough is available for various devices such as PCI, VGA, GPU, HDD, and NIC. Since transparent passthrough in VSBypass is used with nested virtualization, it has two advantages over traditional one. First, it enables device passthrough without configuring user VMs. Second, it can make all user VMs access the same type of virtual devices using passthrough. This is because the targets of passthrough are not the limited number of physical devices but unlimited virtual devices.

BitVisor [24] provides a mechanism called para-passthrough and the hypervisor can intercept part of device access in a VM only when necessary. Using this mechanism, BitVisor enables security features such as disk and network encryption. However, the hypervisor has to be trusted to guarantee the security. Transparent passthrough in VSBypass can add security feature in the cloud hypervisor without relying on the guest hypervisor in the virtualized system.

CloudVisor [31] introduces the security monitor below the virtualized system using nested virtualization and protects user VMs in the virtualized system from untrusted cloud operators. Like VSBypass, it assumes that untrusted cloud operators manage the hypervisor and virtual devices in the virtualized system. It can restrict access to VMs' memory by cloud operators. In addition, it can prevent information leakage and tampering by encrypting memory and disk and checking the integrity. However, CloudVisor does not protect virtual devices used for out-of-band remote management. VSBypass is orthogonal to CloudVisor and can be integrated with CloudVisor.

V-Met [17] enables intrusion detection systems (IDSes) to be offloaded outside the virtualized system using nested virtualization. It prevents IDSes from being compromised by untrusted cloud operators and allows even such cloud operators to manage the entire virtualized system. It provides deep VM introspection to obtain the internal state of user VMs. VSBypass uses several techniques proposed for V-Met, e.g., an ultracall.

FBCrypt [7] and SCCrypt [13] prevent information leakage in out-of-band remote management. I/O data is encrypted between a remote management client and a trusted hypervisor. FBCrypt also checks the integrity of input data using message authentication code. In VSBypass, such encryption and integrity check are not necessary because I/O processing is not performed in virtual devices of the untrusted virtualized system. These systems support para-virtualized I/O devices as well as fully virtualized ones by modifying the virtualized system. VSBypass supports only fully virtualized I/O devices to intercept I/O of user VMs without relying on the virtualized system.

VMware vSphere [29] runs virtual devices and a VNC server in the hypervisor. If the hypervisor is trusted, information leakage from virtual devices is prevented. Otherwise, I/O data of out-of-band remote management can leak. In VSBypass, the hypervisor in the virtualized system does not process such data at all.

SSC [5] prevents information leakage from virtual devices using service domains (SDs), which are provided by a trusted hypervisor. Although virtual devices still run in the untrusted system-wide management VM, SDs can securely encrypt I/O data of user VMs and pass it to untrusted virtual devices. This mechanism can be applied to virtual devices used for out-of-band remote management, but there are the same issues as FBCrypt and SCCrypt. Since SSC establishes an SSL channel between a client and a trusted per-client management VM, users may perform secure out-of-band remote management if they can run appropriate virtual devices in SDs. However, the TCB for SSC is relatively large. It includes the hypervisor and the VM called domB for creating VMs as the system-level TCB and per-client management VMs called Udom0 and SDs as the client-level TCB.

7 CONCLUSION

This paper proposed VSBypass for enabling secure out-of-band remote management outside the virtualized system using transparent passthrough. VSBypass runs the entire virtualized system in the cloud VM using nested virtualization. It intercepts I/O requests of user VMs outside the virtualized system and processes them in shadow devices. Thus, untrusted cloud operators in the virtualized system cannot eavesdrop on or tamper with I/O data of out-of-band remote management. We have implemented transparent passthrough for virtual serial devices, keyboards, mice, and video cards. We confirmed that information leakage from virtual devices was prevented and that the performance of a virtual serial console was comparable to that in the traditional out-of-band remote management.

One of our future work is to dynamically create a proxy VM when a user VM is booted. In the current implementation, we create a proxy VM manually in advance. We need a mechanism for synchronizing the creation of a proxy VM with the boot of a user

VM. Another direction is to support the migration of user VMs. In the current implementation, after a user VM is migrated, the state of shadow devices is not transferred to the destination. As a result, out-of-band remote management is not continued correctly. We are currently developing a mechanism for securely saving and loading the state of shadow devices from the migration tool inside the virtualized system.

ACKNOWLEDGMENT

This work was partially supported by JSPS KAKENHI Grant Number JP16K00101.

REFERENCES

[1] ARM Ltd. 2017. *ARM Architecture Reference Manual – ARMv8, for ARMv8-A Architecture Profile.*
[2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. 2003. Xen and the Art of Virtualization. In *Proc. ACM Symp. Operating Systems Principles.* 164–177.
[3] F. Bellard. [n. d.]. QEMU. https://www.qemu.org/.
[4] M. Ben-Yehuda, M. D. Day, Z. Dubitzky, M. Factor, N. Har'El, A. Gordon, A. Liguori, O. Wasserman, and B.-A. Yassour. 2010. The Turtles Project: Design and Implementation of Nested Virtualization. In *Proc. USENIX Symp. Operating Systems Design and Implementation.* 423–436.
[5] S. Butt, H. A. Lagar-Cavilla, A. Srivastava, and V. Ganapathy. 2012. Self-service Cloud Computing. In *Proc. ACM Conf. Computer and Communications Security.* 253–264.
[6] CyberArk Software. 2009. Global IT Security Service.
[7] T. Egawa, N. Nishimura, and K. Kourai. 2012. Dependable and Secure Remote Management in IaaS Clouds. In *Proc. IEEE Intl. Conf. Cloud Computing Technology and Science.* 411–418.
[8] T. Garfinkel and M. Rosenblum. 2003. A Virtual Machine Introspection Based Architecture for Intrusion Detection. In *Proc. Network and Distributed Systems Security Symp.* 191–206.
[9] IBM Corporation. 2018. IBM Domino 9.0.1 Social Edition Documentation. https://www.ibm.com/support/knowledgecenter/en/SSKTMJ_9.0.1/admin/conf_restrictingadministratoraccess_t.html.
[10] Intel Corp. 2013. 4th Generation Intel Core vPro Processors with Intel VMCS Shadowing.
[11] S. Kawahara and K. Kourai. 2014. The Continuity of Out-of-band Remote Management across Virtual Machine Migration in Clouds. In *Proceedings of the 7th IEEE/ACM International Conference on Utility and Cloud Computing.* 176–185.
[12] K. Kourai and K. Juda. 2016. Secure Offloading of Legacy IDSes Using Remote VM Introspection in Semi-trusted Clouds. In *Proc. IEEE Int. Conf. Cloud Computing.* 43–50.
[13] K. Kourai and T. Kajiwara. 2015. Secure Out-of-band Remote Management Using Encrypted Virtual Serial Consoles in IaaS Clouds. In *Proc. IEEE Int. Conf. Trust, Security and Privacy in Computing and Communications.* 443–450.
[14] C. Li, A. Raghunathan, and N. K. Jha. 2010. Secure Virtual Machine Execution under an Untrusted Management OS. In *Proc. IEEE Int. Conf. Cloud Computing.* 172–179.
[15] C. Li, A. Raghunathan, and N. K. Jha. 2012. A Trusted Virtual Machine in an Untrusted Management Environment. *IEEE Trans. Services Computing* 5, 4 (2012), 472–483.
[16] J. T. Lim, C. Dall, S. Li, J. Nieh, and M. Zyngier. 2017. NEVE: Nested Virtualization Extensions for ARM. In *Proc. ACM Symp. Operating Systems Principles.* 201–217.
[17] S. Miyama and K. Kourai. 2017. Secure IDS Offloading with Nested Virtualization and Deep VM Introspection. In *Proc. European Symp. Research in Computer Security, Part II.* 305–323.
[18] Oracle Corporation. 2018. Oracle Database 2 Day DBA. https://docs.oracle.com/en/database/oracle/oracle-database/18/admqs/administering-user-accounts-and-security.html.
[19] G. Pék, L. Buttyán, and B. Bencsáth. 2013. A Survey of Security Issues in Hardware Virtualization. *Comput. Surveys* 45, 3 (2013), 40:1–40:34.
[20] PwC. 2014. US Cybercrime: Rising Risks, Reduced Readiness.
[21] Red Hat, Inc. [n. d.]. Kernel Based Virtual Machine. http://www.linux-kvm.org/.
[22] N. Santos, K. P. Gummadi, and R. Rodrigues. 2009. Towards Trusted Cloud Computing. In *Proc. USENIX Workshop on Hot Topics in Cloud Computing.*
[23] D. Sgandurra and E. Lupu. 2016. Evolution of Attacks, Threat Models, and Solutions for Virtualized Systems. *Comput. Surveys* 48, 3 (2016), 46:1–46:38.
[24] T. Shinagawa, H. Eiraku, K. Tanimoto, K. Omote, S. Hasegawa, T. Horie, M. Hirano, K. Kourai, Y. Oyama, E. Kawai, K. Kono, S. Chiba, Y. Shinjo, and K. Kato. 2009. BitVisor: A Thin Hypervisor for Enforcing I/O Device Security. In *Proc. ACM SIGPLAN/SIGOPS Int. Conf. Virtual Execution Environments.* 121–130.
[25] H. Tadokoro, K. Kourai, and S. Chiba. 2012. Preventing Information Leakage from Virtual Machines' Memory in IaaS Clouds. *IPSJ Online Trans.* 5 (2012), 156–166.
[26] C. Tan, Y. Xia, H. Chen, and B. Zang. 2012. TinyChecker: Transparent Protection of VMs against Hypervisor Failures with Nested Virtualization. In *Proc. IEEE/IFIP Int. Workshop on Dependability of Clouds, Data Centers and Virtual Machine Technology.*
[27] TechSpot News. 2010. Google Fired Employees for Breaching User Privacy. http://www.techspot.com/news/40280-google-fired-employees-for-breaching-user-privacy.html.
[28] TightVNC Group. [n. d.]. TightVNC. http://www.tightvnc.com/.
[29] VMware Inc. [n. d.]. VMware vSphere Hypervisor. http://www.vmware.com/.
[30] D. Williams, H. Jamjoom, and H. Weatherspoon. 2012. The Xen-Blanket: Virtualize Once, Run Everywhere. In *Proc. ACM European Conf. Computer Systems.* 113–126.
[31] F. Zhang, J. Chen, H. Chen, and B. Zang. 2011. CloudVisor: Retrofitting Protection of Virtual Machines in Multi-tenant Cloud with Nested Virtualization. In *Proc. ACM Symp. Operating Systems Principles.* 203–216.