UVBond: Strong User Binding to VMs for Secure Remote Management in Semi-Trusted Clouds

Keisuke Inokuchi
Kyushu Institute of Technology
inokuchi@ksl.ci.kyutech.ac.jp

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

***Abstract*—In Infrastructure-as-a-Service (IaaS) clouds, remote users access provided virtual machines (VMs) via the management server. The management server is managed by cloud operators, but not all the cloud operators are trusted in semi-trusted clouds. They can execute arbitrary management commands to users' VMs and redirect users' commands to malicious VMs, which is called the *VM redirection attack*. The root cause is that the binding of users to VMs is weak. In other words, it is difficult to enforce the execution of only users' management commands to their VMs. In this paper, we propose *UVBond* for strongly binding users to their VMs to solve this problem. UVBond boots user's VM by decrypting its encrypted disk *inside* the trusted hypervisor. Then it issues a *VM descriptor* to securely identify that VM. To bridge the semantic gap between high-level management commands and low-level hypercalls, UVBond uses *hypercall automata*, which accept the sequences of hypercalls issued by commands. We have implemented UVBond in Xen and confirmed that a VM descriptor and hypercall automata prevented attacks and that the overhead was not large.**

***Index Terms*—virtual machines, clouds, remote management, hypercall automata, disk encryption**

1. Introduction

Infrastructure-as-a-Service (IaaS) clouds provide users with virtual machines (VMs). Users can install their own operating system and applications as they like. They manage provided VMs from remote hosts via the web interface or API. When they perform remote management of their VMs, they first connect to the management server provided in a cloud and then access their VMs via the server. For example, the management server has the ability for booting new VMs, shutting down running VMs, and migrating VMs to other hosts. In addition, users can log in VMs using a feature called out-of-band remote management, which allows users to access virtual serial and graphical consoles without servers running in the VMs.

Although the management server is managed by cloud operators, not all the operators are trusted in semi-trusted clouds [1]–[6]. By semi-trusted clouds, we mean that their providers are trusted but some of the cloud operators may be untrusted. Untrusted cloud operators can abuse the privileges of the management server and mount attacks against users' VMs. They can execute arbitrary management commands to VMs and eavesdrop on and tamper with their sensitive information. In addition, they can redirect users' commands to malicious VMs. Using this *VM redirection attack*, they can steal users' console input inside the malicious VMs. The root cause of these attacks is that the binding of users to their VMs is weak. It is difficult to enforce the execution of only users' management commands to their VMs.

In this paper, we propose UVBond, which strongly binds users to their VMs via encrypted disks of VMs. In UVBond, the trusted computing base (TCB) includes only the hypervisor and hardware. UVBond boots user's VM by decrypting its encrypted disk *inside* the trusted hypervisor and issues a *VM descriptor* to securely identify that VM. Using this descriptor, UVBond guarantees that management commands specified by the user are executed only to the user's VM. Untrusted cloud operators cannot execute commands to users' VMs. They cannot redirect users' access to their malicious VMs. To control the execution of management commands only in the hypervisor, UVBond uses *hypercall automata*, which accept the sequences of hypercalls issued to the hypervisor by users' commands. As long as a hypercall sequence is not rejected, UVBond permits user's access to the VM corresponding to a VM descriptor.

We have implemented UVBond in Xen 4.4.0 [7]. UVBond encrypts and decrypts virtual disks of VMs only in the hypervisor. It supports paravirtual disk drivers, which are mandatory for efficient disk access but difficult to handle only in the hypervisor. To distinguish multiple management commands executed simultaneously, UVBond identifies each process in the hypervisor and applies one hypercall automaton to one process. It also supports secure VM resumption and migration. According to our experiments, it was confirmed that cloud operators could not execute management commands to user's VM or redirect user's commands to a malicious VM. In addition, it was shown that the overhead of using hypercall automata was negligible and the degradation of disk performance was up to 9.5%.

The organization of this paper is as follows. Section 2 describes the VM management in semi-trusted clouds. Section 3 proposes UVBond for strongly binding users to their VMs and Section 4 explains the implementation details.

Section 5 shows our experimental results with UVBond. Section 6 describes related work and Section 7 concludes this paper.

2. VM Management in Semi-trusted Clouds

In IaaS clouds, users manage their VMs via the management server, which is part of the cloud management system. A user first sends a management command with a VM name and other parameters to the management server. Then the server communicates with the agent running in one of the compute nodes and the agent executes privileged operations to the virtualized system, especially the hypervisor. Hereafter, we regard the agents in compute nodes as part of the management server. For example, the management server can boot new VMs, shut down running VMs, and migrate VMs to other hosts. Through the management server, users can log in VMs using out-of-band remote management, which enables indirectly managing VMs via their virtual serial and graphical consoles.

The management server is managed by cloud operators, but not all the operators are trusted in semi-trusted clouds. Semi-trusted clouds are provided by reputable cloud providers and are basically trusted. However, since they hire many operators for daily management, it is difficult to guarantee that all of them are trusted. In fact, it is reported that 28% of cybercrimes are caused by insiders [8]. Malicious system administrators attack systems actively. For example, a site reliability engineer in Google violated user's privacy [9]. In addition, curious but honest system administrators may eavesdrop on attractive information that they can easily obtain from VMs. It is revealed that 35% of system administrators have accessed sensitive information without authorization [10].

Such untrusted cloud operators can abuse the management server or its privileges and attack users' VMs. This is because the binding of users to their VMs is weak. First, cloud operators can execute arbitrary management commands to VMs. For example, they can send magic system requests to VMs, which emulates pressing magic SysRq keys. Then they can show all the register values, which may contain sensitive information, to their serial consoles. Using a technique called VM introspection (VMI) [11], cloud operators can eavesdrop on sensitive information in the memory and disks of VMs. Similarly, they can tamper with the memory and disks. In addition, they can eavesdrop on and tamper with console input and output of out-of-band remote management.

Second, cloud operators can redirect users' management commands to their malicious VMs. We call this attack the *VM redirection attack*. As illustrated in Fig. 1, the VM redirection attack changes a VM accessed by a user in the management server. To eavesdrop on sensitive information, cloud operators create a malicious VM in which malware is installed and execute user's command to the VM. For example, they can steal login passwords in out-of-band remote management by using a malicious login program or a key logger. Since these malicious activities are done inside

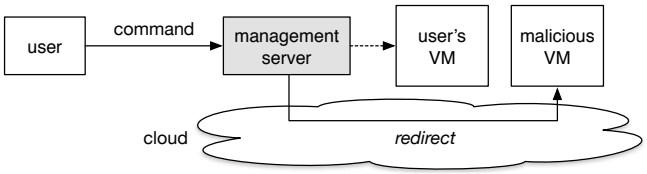


Figure 1: The VM redirection attack.

VMs, they are difficult to prevent even if console input and output are encrypted between users and the hypervisor [12], [13]. In addition, the VM redirection attack can be used for preventing users from detecting malicious activities in VMs using VMI. If cloud operators prepare a VM with a legitimate memory image, users are fooled as normal even when their VMs are compromised.

3. UVBond

3.1. Threat Model

We assume that only the hypervisor and hardware are the trusted computing base (TCB). To trust hardware, we assume that cloud providers themselves are trusted. This assumption is widely accepted [1]–[6] because bad reputations are critical for them. The trustworthiness of the hypervisor can be confirmed by various techniques. For example, remote attestation with TPM guarantees that the hypervisor is booted correctly. Security checks with the system management mode (SMM) [14]–[16] or AMD SVM and Intel TXT [17] can detect attacks against the hypervisor at runtime.

We do not trust cloud operators or privileged components managed by them. Such privileged components include the management server. Since the management server is often run in a privileged VM, which is called Dom0 in Xen, we do not trust the entire privileged VM running on top of the hypervisor. We assume that the privileged VM can be abused by untrusted cloud operators. For example, cloud operators can compromise not only the management server but also the operating system and device emulators running for user's VMs in the privileged VM. However, they cannot attack the hypervisor and hardware. In this paper, we focus on information leakage and tampering, but DoS attacks are out of the scope of this paper.

3.2. Strong Binding of Users to VMs

UVBond strongly binds users to their VMs via encrypted disks of the VMs. It uses disk encryption performed in the trusted hypervisor, instead of traditional disk encryption inside each VM. First, a user securely shares his disk encryption key with the hypervisor at the boot time of his VM. Then the hypervisor associates the key with the VM. Using the registered disk encryption key, UVBond boots the VM by decrypting its encrypted disk inside the hypervisor, as illustrated in Fig. 2. Since the VM cannot be correctly

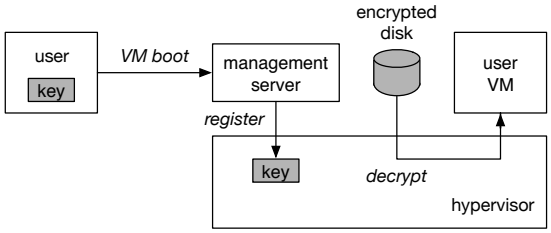


Figure 2: Booting a VM using an encrypted disk.

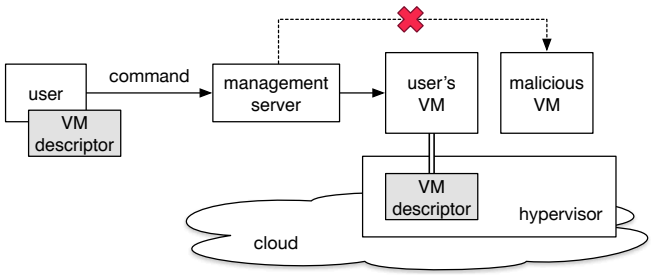


Figure 3: Command execution with a VM descriptor.

booted using the other virtual disks that do not correspond to the key, it is guaranteed that user's own VM is certainly booted. Note that cloud operators can still boot their VM with a malicious disk and its encryption key as user's VM. To prevent this attack, UVBond enables the user to confirm that his disk encryption key is correctly registered.

After the boot of user's VM, UVBond issues a VM descriptor to the user. The descriptor is associated with the VM inside the hypervisor. It is encrypted by the hypervisor and is sent to the user. The user specifies the descriptor when he executes management commands to his VM. On the basis of the descriptor, the hypervisor determines whether privileged operations can be executed to the VM corresponding to the descriptor. In particular, a privileged operation via the hypervisor is called a hypercall. Using the descriptor, UVBond can prevent cloud operators from accessing users' VMs. Only the user that has the descriptor is permitted to access his VM. As illustrated in Fig. 3, UVBond can also prevent the VM redirection attack, which forces a user to access a malicious VM. A user can always access the VM that matches the descriptor.

To prevent information leakage and tampering, it is required to combine UVBond with other techniques. Although only the trusted hypervisor can control access to VMs in UVBond, the access control of hypercalls is not sufficient. For VMI, only a user can map the memory of his VM onto a process using hypercalls, but cloud operators can access the process memory illegally. Therefore it is necessary to obtain encrypted memory data of a VM from the trusted hypervisor and send it to trusted remote hosts using RemoteTrans [5]. Note that the disk of a VM is always encrypted in UVBond. For out-of-band remote management, cloud operators can eavesdrop on and tamper with console input and output via virtual devices of VMs

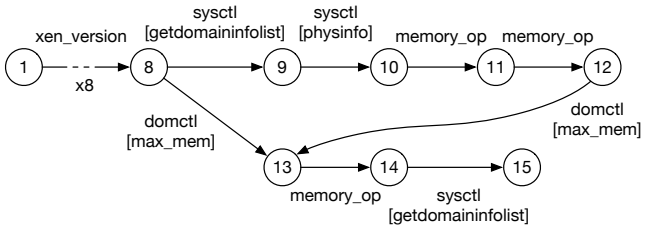


Figure 4: The hypercall automaton for the memory configuration command.

without issuing hypercalls. FBCrypt [12] and SCCrypt [13] should be used to encrypt the data between the remote user and the hypervisor. As a result, cloud operators can obtain only encrypted data from virtual devices. Since UVBond prevents the registration of that encryption key from being redirected to a malicious VM, cloud operators cannot obtain decrypted data inside that VM.

3.3. Hypercall Automaton

A VM descriptor should be able to control the execution of each management command separately, but this is not easy. UVBond assumes that only one management server is shared among all the users and cloud operators, as a traditional system architecture. If each command is exactly equivalent to one hypercall, a user can pass a pair of a command and a VM descriptor to the hypervisor. Then the hypervisor can securely execute the hypercall to the VM corresponding to the descriptor. However, each command usually consists of a set of hypercalls and the other tasks that cannot be executed inside the hypervisor. Since the hypervisor can recognize only hypercalls, it is difficult to securely associate a VM descriptor with the execution of each command.

To bridge this semantic gap, UVBond identifies each management command by a sequence of hypercalls. For each command, it creates a finite state automaton to accept all the sequences of hypercalls issued by the command in advance. This is called a *hypercall automaton*. In general, states in a hypercall automaton have multiple transitions. For example, the command for configuring the memory size of a VM has different hypercall sequences between the first and the following invocations. Therefore its hypercall automaton is created as illustrated in Fig. 4. Note that the input is a hypercall and, if any, its sub-command. When a user accesses his VM, he sends a hypercall automaton as well as the corresponding command and a VM descriptor. The hypervisor permits access to the VM corresponding to the descriptor as long as a hypercall sequence issued by the command is not rejected by the hypercall automaton.

Even if a hypercall sequence is accepted by the specified hypercall automaton, the management command intended by a user is not always executed. Cloud operators might be able to execute another command whose hypercall sequence is accepted by the hypercall automaton but whose behavior

is different. However, commands that generate the same hypercall sequence essentially access a VM in the same manner because a VM can be managed only via hypercalls. Even if an executed command is different from one specified by the user, those commands can be considered as the same in terms of VM management. It may be still possible for attackers to specially craft malicious commands with legitimate hypercall sequences, but hypercall automata at least can make it more difficult to execute malicious commands and raise the bar for attacks.

Using hypercall automata, UVBond can permit some of the management commands even to cloud operators. If only a user has to manage his VMs completely, his burden would become too large. For example, it would be desirable that cloud operators can migrate VMs when the host running the VMs is maintained. To allow cloud operators to execute management commands without a VM descriptor, a user registers the corresponding hypercall automata to his VMs in advance. While the commands corresponding to the registered hypercall automata are executed, even cloud operators can access the specified VMs. Users can determine permitted commands at their discretion and take a trade-off between ease of management and security. Note that secure VM migration requires the memory encryption of VMs using Secure Runtime Environment [2], [18] or VMCrypt [19].

4. Implementation

We have implemented UVBond in Xen 4.4.0 [7]. In Xen, the management server runs in a privileged VM called Dom0 and cloud operators manage users' VMs in Dom0. We have ported AES and RSA in wolfSSL [20] to use them in the hypervisor. We have also developed a management client for UVBond using OpenSSL.

4.1. Secure VM Management

When a new VM is created, an AES key for disk encryption is created and the disk image of the VM is encrypted using the key in the management client. The encrypted disk image is uploaded to a cloud and is stored in Dom0 as usual.

Whenever the VM is booted, the management client generates an AES session key and encrypts it and the disk encryption key using the RSA public key of the hypervisor. The public key is obtained from a trusted key server or in the form of a digital certificate from the management server. Then the management client sends the boot command with the encrypted keys to Dom0. The management server in Dom0 issues a new hypercall for key registration and passes the encrypted keys to the hypervisor. The hypervisor decrypts the passed keys using its own RSA private key and registers the decrypted keys to a being booted VM. Cloud operators cannot decrypt the keys.

According to the boot command, the management server boots a VM with the encrypted disk specified by the user. When the VM accesses its virtual disk, the hypervisor intercepts that access and then decrypts data to be read or encrypts data to be written using the registered disk encryption key. We describe this implementation details in Section 4.2 and Section 4.3. At the same time, the management client checks the correctness of the keys registered to the hypervisor. The details are described in Section 4.4.

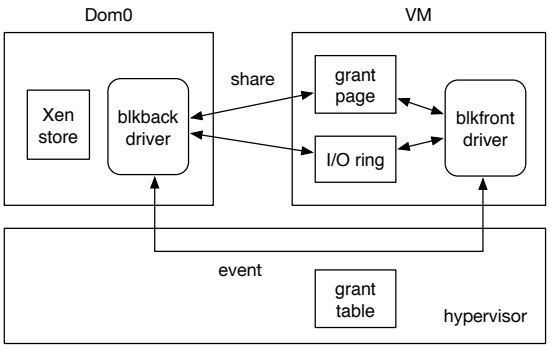


Figure 5: Xen's split device model.

After the boot of the VM, the management server issues a new hypercall and obtains a VM descriptor for the VM from the hypervisor. This descriptor is encrypted with the registered session key. The management server sends the encrypted descriptor back to the client and then the client decrypts it using the session key. To execute a management command, the client encrypts a pair of this descriptor and the hypercall automaton corresponding to the command using the session key and sends the encrypted pair to the server. After the management server completes executing the command, it obtains the result of transitions encrypted by the session key and returns it to the client. We describe the details in Section 4.5.

4.2. Encryption of Para-Virtualized Disk I/O

To access virtual disks of VMs, paravirtual disk drivers are often used. This is because the disk performance is largely improved, compared with using fully virtualized disk drivers. Although Xen supports both para-virtualized and fully virtualized operating systems, Linux uses the paravirtual disk driver by default even in full virtualization. Therefore, the hypervisor has to support disk encryption in para-virtualization. However, this is not easy because the hypervisor cannot trap all accesses to virtual disks.

4.2.1. Traditional Disk I/O. Xen uses the split device model, as illustrated in Fig 5. The paravirtual disk driver consists of the front-end driver called *blkfront* running in a VM and the back-end driver called *blkback* running in Dom0. These drivers share the memory region used for a ring buffer called an *I/O ring*. They communicate with each other using the I/O ring and a signaling mechanism called an *event channel*. When a VM performs disk I/O, the blkfront driver writes a request to the I/O ring and sends an event to the blkback driver via the event channel. When the blkback driver receives that event, it obtains the request from the I/O ring and accesses the disk image of the VM. Upon disk read, it reads data from the disk image and writes the data

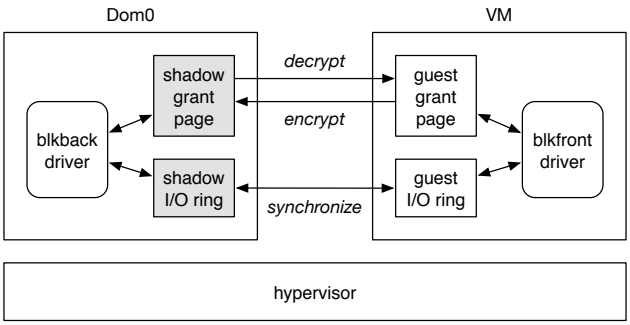


Figure 6: The duplication of grant pages.

to the specified memory page in the VM. Upon disk write, it reads data from the specified memory page in the VM and writes the data to the disk image. Then the blkback driver writes a response to the I/O ring and sends an event to the blkfront driver in the VM.

To share memory pages between Dom0 and a VM, a mechanism called a *grant table* is used in Xen. A VM first registers pages that it intends to share to its grant table. These pages are called *grant pages*. A *grant reference* is assigned to each grant page and Dom0 can map and access a grant page by specifying that grant reference. The grant reference of the page used for the I/O ring is passed from the blkfront driver to the blkback driver via the database called *XenStore* in Dom0. For grant pages used for the buffer of disk I/O, their grant references are passed via requests written to the I/O ring.

4.2.2. Duplication of Grant Pages. To prevent untrusted cloud operators in Dom0 from eavesdropping on data in grant pages after decryption and before encryption by the hypervisor, UVBond duplicates grant pages used for the I/O buffer, as depicted in Fig. 6. For each grant page, it provides an unencrypted page to a VM and an encrypted page to Dom0. A page provided to a VM is called a *guest grant page* and one provided to Dom0 is called a *shadow grant page*. Data in a guest grant page is encrypted and written to the corresponding shadow grant page, while that in a shadow grant page is decrypted and written to the corresponding guest grant page.

UVBond also duplicates the grant page used for the I/O ring but does not encrypt its shadow one. The I/O ring provided to a VM is called a *guest I/O ring*, whereas that provided to Dom0 is called a *shadow I/O ring*. This duplication is used only for synchronization between the guest and shadow I/O rings. The hypervisor copies a request from the guest I/O ring to the shadow one after it completes encrypting data in guest grant pages. Conversely, the hypervisor copies a response from the shadow I/O ring to the guest one after it completes decrypting data in shadow grant pages.

For compatibility with the original Xen, UVBond enables Dom0 to access a shadow grant page using the grant reference of the corresponding guest grant page. For a VM, it associates a grant reference with a guest grant page as usual. For Dom0, in contrast, it associates the same grant reference with the shadow grant page. Therefore, when Dom0 attempts to map a guest grant page passed from a VM, its shadow grant page is mapped instead. Similarly, when Dom0 attempts to unmap the guest grant page, its shadow grant page is unmapped. This gives an illusion to Dom0 as if Dom0 can access a guest grant page.

4.2.3. Disk I/O in UVBond. To perform encryption and decryption of disk I/O, the hypervisor identifies the page used for the I/O ring and creates a shadow I/O ring at the boot time of a VM. For this purpose, it analyzes the communication between a VM and Dom0. The blkfront driver in a VM registers the grant reference of the page used for the I/O ring to XenStore on initialization. For this registration, it writes a request to a ring buffer called the *XenStore ring* and sends an event to XenStore. UVBond intercepts that event in the hypervisor and obtains the grant reference for the I/O ring. The information on the page used for the XenStore ring is passed to the hypervisor at the boot time of a VM. After the hypervisor creates a shadow I/O ring, it copies the guest I/O ring to the shadow one. In addition, it obtains information on the event channel used by the blkfront driver from the request to XenStore.

When the blkfront driver sends a request to the blkback driver, the hypervisor encrypts the requested data using the disk encryption key if the request is for disk write. Since the hypercall for sending an event is called at that time, the hypercall analyzes the request written to the guest I/O ring. It creates a shadow grant page if there is no such a page corresponding to the grant reference included in the request. Then the hypervisor encrypts data stored in the guest grant page and writes it to the shadow grant page. Finally, it copies the request to the shadow I/O ring.

On the other hand, when the blkback driver sends a response to the blkfront driver, the hypervisor decrypts the returned data if the response is for disk read. Since the response includes no grant reference, the hypervisor saves the corresponding request in advance and obtains a grant reference from it. Then the hypervisor decrypts data in the shadow grant page that corresponds to the grant reference and writes it to the guest grant page. Finally, it copies the response to the guest I/O ring. When the guest grant page is released in a VM, the hypervisor releases the shadow one as well.

4.2.4. Using AES-NI in the Hypervisor. We have ported the AES functions with AES-NI from wolfSSL, but special treatment was required to use AES-NI in the hypervisor. AES-NI is a CPU instruction set for AES to improve the performance of encryption and decryption. It needs to use XMM registers, which causes a hardware exception in the hypervisor. This is because the hypervisor in Xen defers the restoration of the XMM registers on CPU scheduling. When the hypervisor accesses one of the XMM registers and an exception occurs, it restores the XMM registers.

To prevent this exception, UVBond clears the TS bit in the CR0 register just before using AES-NI. This bit is set

to cause an exception when XMM registers are accessed. At the same time, UVBond saves the XMM registers. After the use of AES-NI, it restores the XMM registers.

4.3. Encryption of Fully Virtualized Disk I/O

UVBond supports not only para-virtualized but also fully virtualized disk I/O. Even for the operating systems using paravirtual disk drivers, fully virtualized disk I/O is used during the execution of BIOS, which is used before booting the operating system. When BIOS performs disk read by 512-byte data using the IN instruction, the hypervisor emulates that instruction. The read of 512-byte data causes two traps to the hypervisor when BIOS executes the IN instruction for the first 4-byte data and the repeat of the instruction for the remaining 508-byte data. Since the block length of AES is 16 bytes, UVBond decrypts 512-byte data as a whole on the latter trap. Currently, UVBond supports only programmable I/O (PIO).

4.4. Confirming Registered Keys

UVBond confirms that the disk encryption key and the session key registered to the hypervisor are user's when it returns a VM descriptor to the user. When the management server issues the hypercall for obtaining the VM descriptor for a starting VM, the hypervisor appends the disk encryption key to the VM descriptor and encrypts them using the session key. While the management client receives the encrypted data, it decrypts the data using its own session key. If the disk encryption key is extracted correctly, the user can confirm that both the keys registered to the hypervisor are the same as user's.

4.5. Secure Execution of Management Commands

When a user executes a management command to his VM, the management client sends an encrypted pair of a VM descriptor and a hypercall automaton as well as a command and a target VM name to the server. Since a hypercall automaton is a directed graph, UVBond represents it as one-dimensional array to make encryption and network transfers easy, as illustrated in Fig. 7. This array consists of a set of state information, each of which is a pair of an input for transiting to that state and the states to which the automaton can transit from that state. A state is represented as an array index and each input is a hypercall number and, if necessary, the number of its sub-command. For example, the domctl and sysctl hypercalls need to specify a sub-command because they are collective hypercalls and provide various functions.

The management server translates the received VM name into the ID of a running VM and passes it and the received pair to the hypervisor before the execution of the specified management command. First, the hypervisor decrypts the pair using the session key registered to the specified VM. Then, it compares the decrypted VM descriptor

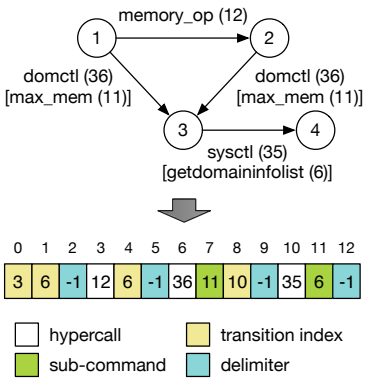


Figure 7: The serialization of a hypercall automaton.

with that registered to the VM. If these are the same, the hypervisor registers the decrypted hypercall automaton to the VM. Otherwise, it considers that as illegal access and returns an error encrypted by the session key. To prevent replay attacks, a monotonic counter can be used between the management client and the hypervisor.

Using the registered hypercall automaton, UVBond checks the validity of issued hypercalls while the command is executed. When a hypercall is issued, the hypervisor examines the states to which the automaton can transit from the current state. The current state is pointed by the index of the array for the hypercall automaton. If the issued hypercall is a possible input, the hypervisor updates the current index and transits to the next state. Then it permits the execution of the hypercall. If the hypercall is not permitted in the hypercall automaton, UVBond considers the issue of that hypercall as illegal and denies the execution. When the execution of the management command is completed, the management server obtains the status of the hypercall automaton from the hypervisor and returns it to the management client. The status is whether the hypercall sequence is accepted or not and is encrypted by the hypervisor. The management client can know whether the command is completed or not.

To distinguish hypercalls issued simultaneously by various management commands, UVBond applies the hypercall automaton only to the process that registers it. Since the hypervisor cannot identify processes of the operating system, UVBond uses the value of the CR3 register, as proposed in [21]. In this register, the physical address of the page directory of a process is stored and is unique to each process. When the hypervisor registers a hypercall automaton to a VM, it associates the current value of this register with the hypercall automaton. When a hypercall is issued, the hypervisor searches for the hypercall automaton on the basis of the value of the CR3 register and uses it, as illustrated in Fig. 8.

4.6. Secure VM Resumption and Migration

UVBond enables users to continue to securely manage their VMs after suspended VMs are resumed. VM suspension saves the states of a VM to a disk, while VM

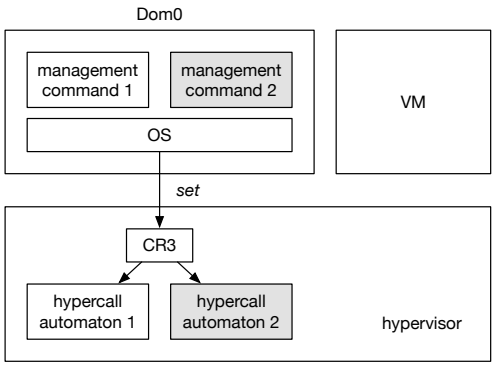
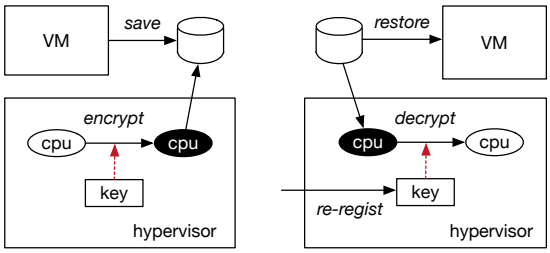


Figure 8: Binding hypercall automatons to management commands.



(a) Suspension (b) Resumption

Figure 9: Secure VM resumption using encrypted CPU state.

resumption restores them. Upon VM resumption, the user registers the disk encryption key of the target VM to the hypervisor again. At this time, cloud operators could register their key and resume the VM with their malicious disk. To prevent this attack, the hypervisor encrypts the CPU state of the VM using the disk encryption key on VM suspension, as illustrated in Fig. 9. Then, it decrypts the state using a newly registered key on VM resumption. If a malicious key is registered, the CPU state cannot be restored correctly and the VM cannot be restarted. Note that the user registers a new session key and receives a new VM descriptor.

For VM migration, UVBond enables VMs to be migrated without explicit key re-registration by users. Unlike VM resumption, a user cannot manually register his disk encryption key to the hypervisor at the destination host because he can send the migration command only to the source host. At the source host, UVBond obtains the disk encryption key and the session key from the hypervisor and transfers them to the destination host, as illustrated in Fig. 10. These keys are encrypted by the public key of the destination hypervisor. To use such a public key by specifying an IP address, UVBond registers pairs of an IP address and a public key to the hypervisor in advance. At the destination host, UVBond automatically registers the disk encryption key and the session key to the hypervisor again. These keys are decrypted by the private key of that hypervisor. Malicious replacement of the keys is prevented by the same mechanism as VM resumption. Note that the user can use the same VM descriptor before VM migration.

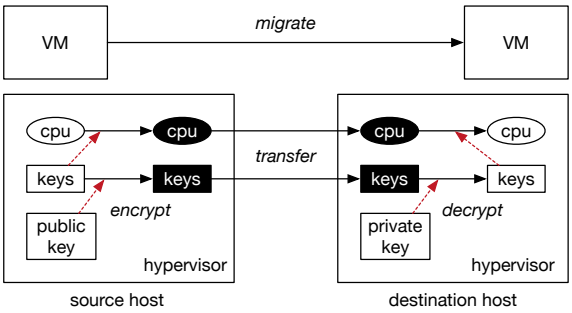


Figure 10: Secure VM migration with keys.

5. Experiments

We conducted experiments to confirm the effectiveness of UVBond. We used a PC with an Intel Xeon E3-1290 v2 processor, 8 GB of memory, and 1 TB of HDD. For a VM, we assigned two virtual CPUs, 2 GB of memory, and 20 GB of virtual disk. We used Xen 4.4.0 modified for UVBond and ran Linux 3.16 in Dom0 and Linux 3.13 in a VM. For comparison, we used vanilla Xen without modification. As a client host or a destination host of VM migration, we used a PC with the same spec and software as the above. These two hosts were connected with Gigabit Ethernet.

5.1. VM Boot

First, we examined that a VM could be booted with a correct disk encryption key. When we registered a correct key to the hypervisor, the VM could boot normally. However, using an incorrect key, the VM could not read the boot loader from its disk because disk data was not decrypted correctly. This means that malicious cloud operators cannot boot users' VMs.

Next, we examined the boot time of a VM to confirm the overhead of UVBond. For comparison, we used UVBond with AES-NI disabled. We measured the time from when we executed the create command for creating a VM until the system was booted in the VM. Since the page cache in Dom0 affected the disk performance of the VM largely, we measured the time both when we cleared the page cache before booting the VM and when we booted the VM again with the page cache kept. As shown in Fig. 11, the boot time in UVBond was 6 seconds longer than that in vanilla Xen without depending on the page cache. This is due to the overhead of additional operations performed by UVBond, including disk encryption. Compared with when we disabled AES-NI, it was shown that AES-NI made the boot only 1 second faster.

5.2. Execution of Management Commands

First, we examined that UVBond could detect illegal commands when we used a VM descriptor and a hypercall

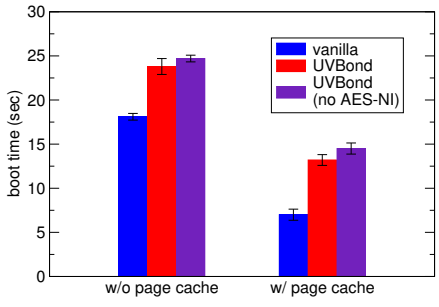


Figure 11: The boot time of a VM.

TABLE 1: The size of hypercall automata used in the experiments.

command	state	transition
pause	9	8
unpause	10	10
mem-set	15	15
shutdown	10	9
destroy	12	13
save	20	21

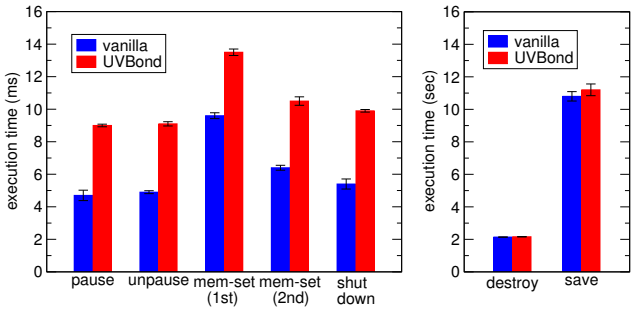
automaton. We executed management commands of Xen's xl tool for pausing a VM (pause) and resuming it (unpause). As a result, we confirmed that management commands could be normally executed only when both the VM descriptor and the hypercall automaton match the target VM and the management command, respectively. When we specified an illegal descriptor that did not correspond to the specified VM or executed an illegal command that did not correspond to the specified hypercall automaton, we could detect that.

Next, we examined the execution time of management commands. We executed six management commands: pause, unpause, mem-set for changing the memory size of a VM, shutdown for shutting it down, destroy for destroying it, and save for suspending it. For the mem-set command, we measured the time of the first and second execution because the hypercall sequences for them were different. The size of the hypercall automata used for these commands is shown in Table 1. The save command had the most complex hypercall automaton.

Fig. 12 shows the execution time. For short-time commands, the overhead of UVBond was 4 ms, which included the registration and runtime check of a hypercall automaton. In addition to the execution time, it took 37 ms to send a VM descriptor and a hypercall automaton to the management server in UVBond. Although this is much longer than the execution time itself, this overhead is almost negligible because these commands are not executed frequently. For long-time commands, the overhead of UVBond was negligible because the number of issued hypercalls was relatively small.

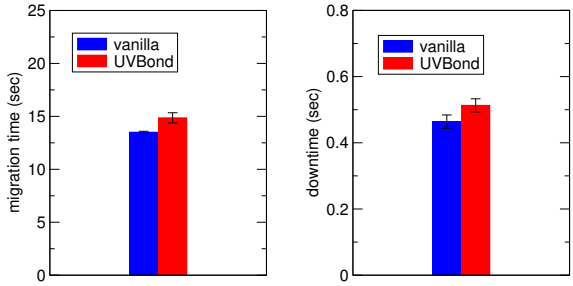
5.3. VM Migration

First, we examined that a VM could be migrated only with a correct disk encryption key. When the source host transferred a correct key to the destination, a migrated



(a) Short-time (b) Long-time

Figure 12: The execution time of management commands.



(a) Migration time (b) Downtime

Figure 13: Migration performance.

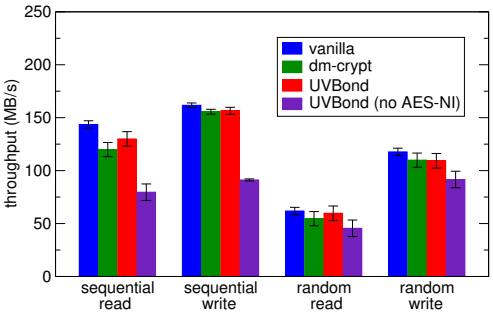
VM was restarted correctly. However, when we discarded the transferred key and registered an incorrect key at the destination host, we could not access a migrated VM. This is because the CPU state of the VM was not restored correctly.

Next, we examined the performance of migrating a VM using UVBond. We measured the migration time, which was the time from when we executed the migrate command until it was completed. As shown in Fig. 13(a), the migration time in UVBond was 1.4 seconds longer than vanilla Xen because UVBond had to transfer the disk encryption key and the session key and encrypt the CPU state. Also, we measured the downtime, which was the time from when the VM was paused at the source host until it was resumed at the destination host. As shown in Fig. 13(b), the downtime in UVBond became only 50 ms longer.

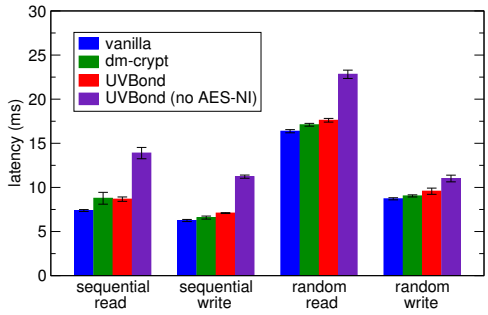
5.4. Disk I/O Performance

We examined the disk I/O performance using the fio benchmark [22]. In addition to UVBond with or without AES-NI and vanilla Xen, we used the system using Linux dm-crypt [23] inside a VM. dm-crypt encrypts and decrypts disks in the operating system of a VM. For dm-crypt, we used AES-ECB as the encryption method, which was the same as that of UVBond. We measured read and write performance of sequential and random access.

Fig. 14 shows the throughput and the latency when a VM used a virtual disk on a local disk. Compared with vanilla Xen, the throughput in UVBond degraded only by 3-10% and the latency increased only by 0.8-1.3 ms thanks to

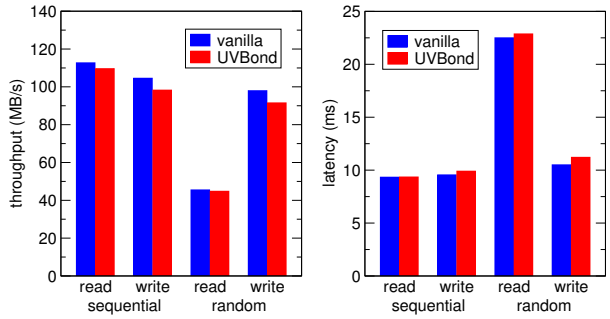


(a) Throughput



(b) Latency

Figure 14: The performance of file access.



(a) Throughput (b) Latency

Figure 15: The performance of file access on NFS.

AES-NI. Without AES-NI, the throughput degradation was 45% and the latency increase was 6.5 ms at maximum. The throughput in UVBond was comparable to or even better than that in dm-crypt, while the latency was slightly longer.

Fig. 15 shows the throughput and the latency when a VM used a virtual disk on NFS. This configuration is often used when a VM is migrated. The performance degradation in UVBond was similar to that when using a local disk. The throughput degraded by 1.6-6.6%, while the latency increased by 0.0-0.7 ms.

6. Related Work

Self-Service Cloud (SSC) [4], [24] can prevent cloud operators from illegally accessing users' VMs. For each user, it provides a dedicated management VM called Udom0, which is not interfered by cloud operators. Since each user's VMs can be managed only via his Udom0, cloud operators cannot eavesdrop on or tamper with the VMs. The disk integrity of Udom0 is verified with vTPM, which runs in a domain builder called domB. The user accesses Udom0 using a management server called a dashboard through an SSL channel, which is securely established at the boot time of Udom0. User's VMs are securely created via domB.

However, the TCB of SSC is quite large because it includes not only the hypervisor and hardware but also several privileged VMs such as Udom0, domB, and a dashboard VM. Udom0 is not a system-level TCB but a client-level TCB, whose compromise affects user's VMs. Udom0 is protected by the trusted hypervisor, but information leakage and tampering can occur if the system inside Udom0 is compromised by exploiting its vulnerabilities. Similarly, the systems inside domB and a dashboard VM can be compromised. The system including the operating system in such privileged VMs has much larger attack surfaces than the hypervisor. The TCB of UVBond is smaller because it does not include any privileged VMs.

Several techniques for secure disk encryption have been proposed. SSC enables users to encrypt the disks of VMs using special VMs called service domains (SDs). Each user can run his own SDs, while cloud operators cannot interfere with user's SDs. Like Udom0, SDs are also a client-level TCB and are protected by the trusted hypervisor. However, once the systems inside SDs are compromised, disk data is leaked or tampered with.

BitVisor [25] can encrypt the disk of a VM using a parapass-through driver in the trusted hypervisor. The hypervisor intercepts only minimum hardware access needed for disk encryption, while the other access passes through the hypervisor. The driver for the ATA host controller supports not only PIO but also DMA transfers by using shadow DMA descriptors and shadow buffers. UVBond can also support DMA transfers using the same technique. Unlike UVBond, BitVisor supports only fully virtualized operating systems and cannot use para-virtual disk drivers in a VM.

CloudVisor [3] encrypts the disks of VMs in the security monitor running below the hypervisor, using nested virtualization [26]. It intercepts I/O requests of VMs and encrypts or decrypts data. In addition, CloudVisor checks the integrity of the disks. It provides necessary hash data to the security monitor via the management VM. Using the hash data, it is guaranteed that VMs are booted properly with the disk images specified by users. CloudVisor can exclude even the hypervisor from the TCB, but the overhead of nested virtualization is not small.

Console input and output in out-of-band remote management are encrypted in the hypervisor using FBCrypt [12] and SCCrypt [13]. Like UVBond, FBCrypt also duplicates VM's framebuffer used for a para-virtualized graphical console and provides an encrypted one to Dom0. However, it supports only memory sharing directly using page frame numbers, not grant references. Since FBCrypt does not duplicate I/O rings unlike UVBond, it has to write keyboard input to an I/O ring using a new hypercall. It is necessary to

prevent the frontend driver in a VM from accessing data that is not yet decrypted by the hypervisor. Therefore, FBCrypt needs to modify the back-end driver in Dom0. UVBond solves this synchronization problem without modifying disk drivers by duplicating I/O rings.

System-call automata [27] are used for intrusion detection systems (IDSes). Such IDSes detect intrusion on the basis of a sequence of system calls issued by a process. They trace the program execution in advance and record normal behavior as a system-call automaton. If a process issues a system call that is not accepted by the automaton, the IDSes detect abnormal behavior. A hypercall automaton used in UVBond is an application to the hypervisor.

7. Conclusion

This paper proposed UVBond for providing strong user binding to VMs. UVBond enables only a user to boot his VM by decrypting its encrypted disk inside the trusted hypervisor. Then it issues a VM descriptor for securely identifying that VM. To bridge the semantic gap between high-level management commands and low-level hypercalls, UVBond uses hypercall automata and accepts only the sequences of hypercalls issued by user's commands. Using UVBond, untrusted cloud operators cannot execute arbitrary commands to user's VMs or redirect user's commands to their malicious VMs. We have implemented UVBond in Xen and confirmed that the overhead was not large.

Our future work was to apply UVBond to large cloud management systems such as OpenStack [28]. To support UVBond in such systems, we have to extract used management commands and create their hypercall automata. Also, we need to modify the Web interface and API so as to send VM descriptors and hypercall automata as well as commands.

Acknowledgment

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

References

[1] N. Santos, K. P. Gummadi, and R. Rodrigues, "Towards Trusted Cloud Computing," in *Proc. Workshop on Hot Topics in Cloud Computing*, 2009.

[2] C. Li, A. Raghunathan, and N. K. Jha, "Secure Virtual Machine Execution under an Untrusted Management OS," in *Proc. IEEE Int. Conf. Cloud Computing*, 2010, pp. 172–179.

[3] F. Zhang, J. Chen, H. Chen, and B. Zang, "CloudVisor: Retrofitting Protection of Virtual Machines in Multi-tenant Cloud with Nested Virtualization," in *Proc. ACM Symp. Operating Systems Principles*, 2011, pp. 203–216.

[4] S. Butt, H. A. Lagar-Cavilla, A. Srivastava, and V. Ganapathy, "Self-service Cloud Computing," in *Proc. ACM Conf. Computer and Communications Security*, 2012, pp. 253–264.

[5] K. Kourai and K. Juda, "Secure Offloading of Legacy IDSes Using Remote VM Introspection in Semi-trusted Clouds," in *Proc. IEEE Int. Conf. Cloud Computing*, 2016, pp. 43–50.

[6] S. Miyama and K. Kourai, "Secure IDS Offloading with Nested Virtualization and Deep VM Introspection," in *Proc. European Symp. Research in Computer Security*, 2017, pp. 305–323.

[7] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the Art of Virtualization," in *Proc. Symp. Operating Systems Principles*, 2003, pp. 164–177.

[8] PwC, "US Cybercrime: Rising Risks, Reduced Readiness," 2014.

[9] TechSpot News, "Google Fired Employees for Breaching User Privacy," http://www.techspot.com/news/40280-google-fired-employees-for-breaching-user-privacy.html, 2010.

[10] CyberArk Software, "Global IT Security Service," 2009.

[11] T. Garfinkel and M. Rosenblum, "A Virtual Machine Introspection Based Architecture for Intrusion Detection," in *Proc. Network and Distributed Systems Security Symp.*, 2003, pp. 191–206.

[12] T. Egawa, N. Nishimura, and K. Kourai, "Dependable and Secure Remote Management in IaaS Clouds," in *Proc. Intl. Conf. Cloud Computing Technology and Science*, 2012, pp. 411–418.

[13] K. Kourai and T. Kajiwara, "Secure Out-of-band Remote Management Using Encrypted Virtual Serial Consoles in IaaS Clouds," in *Proc. Int. Conf. Trust, Security and Privacy in Computing and Communications*, 2015, pp. 443–450.

[14] J. Rutkowska and R. Wojtczuk, "Preventing and Detecting Xen Hypervisor Subversions," Black Hat USA, 2008.

[15] J. Wang, A. Stavrou, and A. Ghosh, "HyperCheck: A Hardware-assisted Integrity Monitor," in *Proc. Int. Symp. Recent Advances in Intrusion Detection*, 2010, pp. 158–177.

[16] A. Azab, P. Ning, Z. Wang, X. Jiang, X. Zhang, and N. Skalsky, "HyperSentry: Enabling Stealthy In-context Measurement of Hypervisor Integrity," in *Proc. ACM Conf. Computer and Communications Security*, 2010, pp. 38–49.

[17] J. McCune, B. Parno, A. Perrig, M. Reiter, and H. Isozaki, "Flicker: An Execution Infrastructure for TCB Minimization," in *Proc. European Conf. Computer Systems*, 2008, pp. 315–328.

[18] C. Li, A. Raghunathan, and N. K. Jha, "A Trusted Virtual Machine in an Untrusted Management Environment," *IEEE Trans. Services Computing*, vol. 5, no. 4, pp. 472–483, 2012.

[19] H. Tadokoro, K. Kourai, and S. Chiba, "Preventing Information Leakage from Virtual Machines' Memory in IaaS Clouds," *IPSJ Online Trans.*, vol. 5, pp. 156–166, 2012.

[20] wolfSSL Inc., "wolfSSL Embedded SSL/TLS Library," https://www.wolfssl.com/.

[21] S. Jones, A. Arpaci-Dusseau, and R. Arpaci-Dusseau, "Antfarm: Tracking Processes in a Virtual Machine Environment," in *Proc. USENIX Annual Technical Conf.*, 2006.

[22] J. Axboe, "fio: Flexible I/O Tester," https://github.com/axboe/fio.

[23] M. Brož, "dm-crypt: Linux Kernel Device-mapper Crypto Target," https://gitlab.com/cryptsetup/cryptsetup/wikis/DMCrypt.

[24] S. Butt, V. Ganapathy, and A. Srivastava, "On the Control Plane of a Self-service Cloud Platform," in *Proc. Symp. Cloud Computing*, 2014.

[25] 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, "BitVisor: A Thin Hypervisor for Enforcing I/O Device Security," in *Proc. Int. Conf. Virtual Execution Environments*, 2009, pp. 121–130.

[26] M. Ben-Yehuda, M. D. Day, Z. Dubitzky, M. Factor, N. Har'El, A. Gordon, A. Liguori, O. Wasserman, and B.-A. Yassour, "The Turtles Project: Design and Implementation of Nested Virtualization," in *Proc. USENIX Symp. Operating Systems Design and Implementation*, 2010, pp. 423–436.

[27] S. Hofmeyr, S. Forrest, and A. Somayaji, "Intrusion Detection Using Sequences of System Calls," *Computer Security*, vol. 6, pp. 151–180, 1998.

[28] The OpenStack Project, "OpenStack Open Source Cloud Computing Software," https://www.openstack.org/.