# TRIOB: A Trusted Virtual Computing Environment Based on Remote I/O Binding Mechanism

Haifeng Fang[1,2], Hui Wang[3], Yiqiang Zhao[1], Yuzhong Sun[1], and Zhiyong Liu[1]

[1] Key Laboratory of Computer System and Architecture, Institute of Computing Technology, Chinese Academy of Sciences
[2] Graduate University of Chinese Academy of Sciences
[3] High Performance Computer Research Center, Institute of Computing Technology, Chinese Academy of Sciences
{fanghaifeng,wanghui}@ncic.ac.cn,{zhaoyiqiang,yuzhongsun,zyliu}@ict.ac.cn

**Abstract.** When visiting cloud computing platforms, users are very concerned about the security of their personal data. Current cloud computing platforms have not provided a virtual computing environment which is fully trusted by users. Meanwhile, the management domain of cloud computing platform is subject to malicious attacks, which can seriously affect the trustworthiness of the virtual computing environment. This paper presents a new approach to build a trusted virtual computing environment in data centers. By means of three innovative technologies, the user's data can be remotely stored into trusted storage resources, the user's virtual computing environment is isolated, and the user can automatically detect the rootkit attacks against the cloud computing management domain. We design and implement a Xen-based prototype system called TRIOB. This manuscript presents the design, implementation, and evaluation of TRIOB, with a focus on rootkits detection.

**Keywords:** virtual computing environment, remote I/O binding, virtual machine isolation, rootkits detection.

## 1 Introduction

Cloud computing has become an increasingly popular computing solution. Using VMM technologies such as Xen[1], and VMware, in the cloud computing platform people can dynamically create a large number of virtual machines (VMs) to meet customers' requirements. For example, through IaaS[2] services (like Amazon's EC2[15]), users can get their own exclusive VMs for running a number of security-sensitive applications. However, from the users' point of view, data security and privacy is still an unmet concern, which is currently the major obstacle to the development of cloud computing[3].

Presently, Xen, an open source virtualization system software, is widely used in data centers. In Xen system, the virtual machine (VM) is called Domain. In order to provide a management entrance for system administrator, after Xen

S.-H. Heng, R.N. Wright, and B.-M. Goi (Eds.): CANS 2010, LNCS 6467, pp. 241–260, 2010.

started, it immediately creates a unique privilege VM (called Domain0) to run the VM management program. Domain0 provides interface for the administrator to create, delete, start and stop multiple user VMs (called DomainU), and allocate resources to the users' VMs. In Xen-based virtual computing environment, the trustworthiness of DomainU depends not only on its internal software configuration, but also on the internal software in Domain0. This is decided by the device driver model of Xen (as illustrated in Fig. 1)[1].

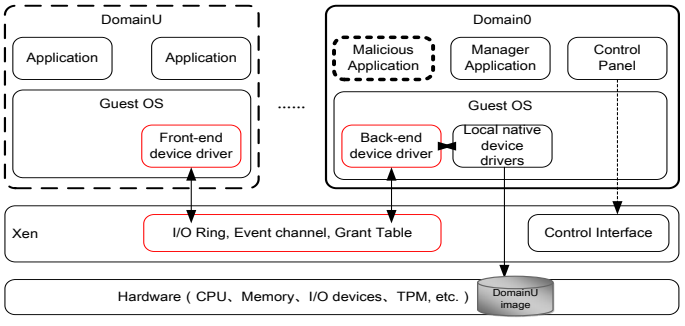


**Fig. 1.** The architecture of Xen and splitting device driver model

However, there are still some security weaknesses in Xen-based virtual computing environment as follows.

First, the code size of the whole trusted computing base (TCB) of the virtual computing environment keeps increasing. To solve this problem, Xen introduces a new domain called Isolated Driver Domain (IDD)[29]. In this way, Domain0 is split away, having some device drivers in Domain0 encapsulated in a separated IDD. This not only reduces the code size of the guest OS kernel in Domain0, but also improves the device driver's reliability. However, in the current Xen architecture, it is very common that cloud providers still take Domain0 as the IDD. Besides, the operating system codes related to the management applications inside Domain0 could not be completely separated away. Consequently, we need to dynamically monitor them to ensure their integrity.

Second, the assumption that Domain0 of each server in data centers is trusted may not always be true. There exist several external or internal weakness in Domain0, e.g., wrong configurations and software bugs, which can compromise the trustworthiness of Domain0. It is common in the current virtualized environment to have a management console (like XenCenter, HyperVM) that manages DomainUs through the control interface (like XenAPI) in Domain0. While these consoles help the administrators to manage the machines, they open new vulnerabilities, e.g., the XenAPI HTTP interface has a cross-site scripting vulnerability[31]. Compromising a management console allows an attacker to control all the VMs managed by it. For instance, on June 8th 2009, 100,000 hosted websites were affected by a zero-day SQL injection hole in the HyperVM 2.09[32]. With this attack, the intruders gained the root privileges to perform malicious operations (like installing rootkits) in Domain0.

Finally, malicious applications may be able to affect the data integrity of DomainU through Domain0. Most of the management tasks are executed in Domain0, including the I/O processing, Domain management, etc. After DomainU is created, Domain0 can read and write the memory space that belongs to DomainU through one management interface (xc_map_foreign_range). Unfortunately, malicious applications can also run in Domain0, as we mentioned. Once these applications control the interface, it will seriously affect the integrity of DomainU. To address this issue, Xen developers introduced one new concept called DomB[13]. With this new addition, the interface can only be called by the DomB in which there only deployed some special programs. Although the interface can be protected by the special Domain, sometimes DomainU need to close down the xc_map_foreign_range interface for security purpose, but it can not do this currently. Therefore, we should re-arrange the privileges amongst Xen, Domain0 and DomainU to meet the users'security requirements.

Currently, through the virtual trusted platform model (TPM)[8], we can establish the trust connection between Domains belonging to different management fields. But the TPM technology can only ensure the Domains' configuration integrity at startup[9][21], and it cannot guarantee the runtime integrity of the processes running in the Domains when they are attacked by malicious programs like rootkits. Existing research work [16][17][18] mainly enhance the trustworthiness of DomainU by means of "out-of-the-box" monitoring technologies. However, because the trustworthiness of DomainU also depends on Domain0, we must ensure that Domain0 is in a trusted running state[34][35][36]. Therefore, no matter how Domain0 was split, the integrity of the processes in Domain0 needs to be monitored, otherwise DomainU will be in an untrustworthy state.

Based on the above analyses, we designed and implemented a new trusted virtual computing environment (called TRIOB) for data center with some new trust-enhancing techniques. In this virtual computing environment, users can specify the resource configuration and the running mode of DomainU in which users' applications are deployed. In detail, by means of the new remote I/O binding technique, DomainU can be bound with the remote storage resources located in the users' physical machines outsides cloud computing platform. During users' applications are running, DomainU can be in a new trusted running mode to absolutely isolate from other DomainUs and Domain0 in data center. Meanwhile, the integrity of the Domain0 is measured dynamically and remotely verified by users.

We have tested the TRIOB prototype system. Our experimental results show that compared to the traditional remote I/O binding technique, the TRIOB's performance is improved. At the same time, under the introduction of the trust-enhancing mechanisms, the TRIOB system can timely detect the rootkit attacks against the Domain0, while the overhead remains in an acceptable range for users.

The rest of this paper is organized as follows: Section 2 describes the background information. Section 3 gives a specific application scenario, and discusses the existing challenges. In Section 4, we give the architecture of prototype system and its work principle. Section 5 introduces an implementation of the prototype

system in detail. Section 6 is the prototype evaluation. Some related works are introduced in Section 7. Finally concluding remarks are presented in Section 8.

## 2 Background

### 2.1 Splitting Device Driver Model

The unique splitting device driver model (illustrated in Fig. 1)[1] is the core feature of Xen. Applications in DomainU access I/O devices through the front-end device driver which provides virtual I/O devices for DomainU, such as virtual block device. Whenever these applications produce I/O operations, the front-end device driver receives these I/O requests. Then these I/O requests are forwarded to the back-end device driver in Domain0 through the communication mechanisms (I/O ring, event channel, grant table) between VMs. After the back-end device driver receives these I/O requests, it checks whether the requests are legitimate and then calls the local native device drivers to perform real I/O operations. When the I/O operations are completed, the back-end device driver will notice the front-end device driver which reports to the applications in DomainU.

With the splitting device driver model, DomainU just works with virtual I/O devices, and it does not need to concern with the physical I/O devices' type and location. For example, a VM can be configured with some virtual CPUs and virtual memory which derive from the local physical node in data center, and virtual storage devices which derive from other non-local physical nodes' storage devices. In this situation, the VM can use local resources for running applications, and transparently store data into the other remote physical nodes. However, Xen currently only supports this idea indirectly based on traditional technologies such as NFS, NBD[10][11], and it does not consider the relevant trustworthiness issues that may arise because of introduction of some remote I/O binding in the current model.

### 2.2 Kernel-Level Rootkit Attacks against Domain0

To ensure the trustworthiness of Domain0, we need to analyze the possible attacks against Domain0. Currently, the guest OS running in Domain0 is XenoLinux (Linux's modified version running on Xen). Various rootkit attacks, especially the kernel-level ones, are the most serious threats for Linux operating system[19]. Unfortunately, due to the back-end device drivers locating within the Linux kernel in Domain0, the kernel-level rootkit attacks against Domain0 will also affect DomainU.

In general, rootkits are collections of programs that enable attackers who have gained administrative control of a host to modify the host's software, usually causing it to hide their presence from the host's genuine administrators[27]. Linux-oriented kernel-level rootkit usually enters into the kernel space through accessing /dev/kmem (or /dev/mem), or it is loaded into kernel space as a kernel module (LKM). After the rootkit enters into the kernel space, it can make damages as follows. 1) Modifying the kernel text code, such as the virtual file system, the device drivers; 2) Modifying the kernel's critical data structures such as the

system call table and the interrupt table. From users' point of view, it is necessary to monitor the integrity of the kernel code, the key kernel data structures and other critical kernel memory space in Domain0, so that users can timely make awareness of the exception in Domain0 and take effective countermeasures.

## 3 Scenario and Challenges

Typically, a trusted computing environment should enable users to know its software configuration and runtime state. In addition, while the computing environment is under damage, it should provide the mechanism to protect users' sensitive data[4].

To build such a Xen-based trusted virtual computing environment, we suppose that the users need to get a VM from the IaaS service in cloud computing platform. But, they do not trust the cloud computing platform, especially its internal storage resources. Meanwhile, they only want to lease the virtual computing environment (that is DomainU in Xen) to run some security-sensitive applications. In this scenario, users can customize the lent DomainU with a special recourse configuration in which its virtual storage device is bound with the users' storage resources. From user's point of view, the actual storage resource remotely locates in user's local physical machine.

At first, we assume that Xen is trustworthy because its code size is very small and the Domain in running state has almost no power to destroy Xen[30]. Note that this assumption is consistent with that of many other VMM-based security research efforts [5][14]. Then, the virtual computing environment for user needs to undergo two stages as follows.

(1) Starting DomainU. User needs to specify the DomainU's resource configuration (like the location information of the remote storage resources) through the service interface of the cloud computing platform. The information will be sent to Domain0 which is responsible for creating the corresponding DomainU. When starting DomainU, user needs to get the configuration metric information of DomainU, Domain0 and Xen to determine whether he can establish a trusted connection with them. By means of TPM technology, we can achieve this goal in that we do not focus on this stage in this paper.
(2) Running user's applications in DomainU. These applications are installed in user's storage resources, so user can believe that DomainU is trustworthy. When these applications access data from user's remote storage devices, the I/O requests in DomainU will be forwarded to the guest OS kernel in Domain0. However, currently user cannot control Domain0 from DomainU. According to the preceding analysis, at this stage, Domain0 should be excluded from the TCB.

Based on the above analysis, the Xen-based virtual computing environment is trustworthy only if it has two basic properties: trusted storage resources and secure runtime computing environment (we call it TRIOB-DomainU). To achieve these goals, we need to overcome the following challenges.

First, how to bind TRIOB-DomainU with the remote user's storage resources? By using NFS technology, we can mount the remote storage image file onto Domain0. But this solution is not secure because Domain0 can directly access the content inside the mounted image file. Meanwhile, it would be better that the remote I/O binding mechanism can match with the splitting device driver model and facilitate with TRIOB-DomainU for trusty purpose. Thus, we need to design a new secure remote I/O binding mechanism for TRIOB-DomainU, by which users only need to provide the location information of the remote image file.

Second, how to allow user to control the runtime state of TRIOB-DomainU? DomainU should be absolutely isolated from other VMs, so that the user believes his applications running in a secure space. As mentioned before, we need to redivide the privileges amongst Xen, Domain0 and the DomainU. For example, after DomainU is created, user can close up the xc_map_foreign_range interface to prohibit Domain0 from mapping the memory of the DomainU. Most importantly, DomainU should own the power of monitoring some memory regions of Domain0 to check its integrity. At the same time, we should not allow attacker to damage Domain0 by means of the privileges of the DomainU. That is to say, DomainU can only specify the location and type of mapped memory area in Domain0, and can only obtain the integrity information related to the memory area. On the other hand, the memory area that can be read by DomainU must be verified by Xen so that some valuable contents in Domain0 are protected by Xen. The most important thing is that the proposed mechanism needs to ensure that only the memory area that mapped to the DomainU can be accessed by the user.

Finally, how to allow user to be aware of the exceptions, e.g., the attacks against VM's kernel, in cloud computing platform? According the security requirements of TRIOB-DomainU, user hopes to perceive Domain0 kernel's exception in time and make the appropriate treatments. Meanwhile, in data center there are many DomainUs serving to different users and the users would take care of different security issues. It is obvious that someone needs to dynamically monitor and measure the memory of Domain0 to check its integrity. But, who should be responsible for this task? Is TRIOB-DomainU or Xen? We believe that Xen is better because in Xen space we can easily monitor any memory area belonging to different Domains including Domain0. Furthermore, who is responsible for the verification task? For the special scenario above, it is the user who should do this work. Especially, user needs to determine whether the I/O data produced from TRIOB-DomainU could be stored into user's storages based on the runtime state of Domain0.

To address the above challenges, we propose a new approach to construct a trusted virtual computing environment and we call it TRIOB system. Its architecture is shown in Fig. 2.

## 4 Architecture of TRIOB

The TRIOB system consists of two parts, the computing environment running in data center and the storage resource locating in the user's physical node. In

Fig. 2, the left side is the computing environment (TRIOB-DomainU) which provides a trusted running environment for user's applications, and the right side is the user's physical machine which provides reliable storage resources. In TRIOB-DomainU, the storage resources are some VM image files (that is, DomainU's image). From the functionality viewpoint, the TRIOB system includes three subsystems, that is, the remote I/O subsystem, the mode-control subsystem and the dynamic monitoring and measurement subsystem. The three subsystems can help overcome the challenges respectively.

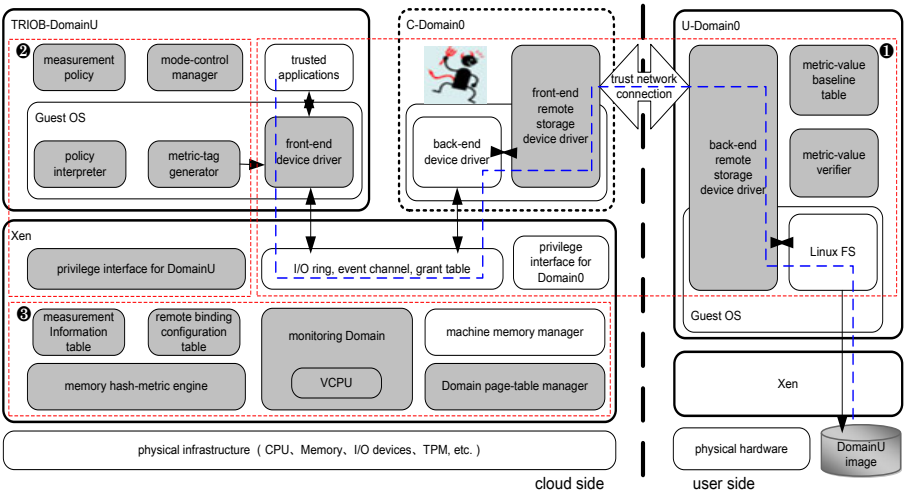


**Fig. 2.** Architecture of TRIOB system ( ①remote I/O subsystem; ②mode-control subsystem; ③dynamic monitoring and measurement subsystem)

The remote I/O subsystem consists of the front-end remote storage device driver, the back-end remote storage device driver, and the associated remote verification modules. The front-end remote storage device driver is responsible for transferring the I/O requests produced by applications to the back-end remote storage device driver through network. If TRIOB-DomainU is in the trusted running mode, each I/O request will be attached with the tag which contains the metric-hash value of C-Domain0. In the user side, when the back-end remote storage device driver receives the I/O requests, it will first send the tag to the metric-value verifier who is responsible for the verification based on the metric-value baseline. Only these I/O data belonging to trusted I/O requests can be stored into the user's local storage resource, that is, the DomainU's image. To enhance the security of the remote I/O channel, we can encrypt the corresponding I/O data in TRIOB-DomainU and decrypt them in user side.

The mode-control subsystem consists of the mode-control manager, the measurement policy, the policy semantic interpreter, the metric-tag generator and the privilege interface for DomainU. When security-sensitive applications are running in TRIOB-DomainU, the mode-control manager can switch DomainU

into the trusted mode through calling the privilege interface. The user can specify the policy what content of kernel memory in C-Domain0 to be measured and how to measure. Then the policy interpreter is responsible for translating the policy and registering this information into the measurement information table and the remote binding configuration table, respectively. The privilege interfaces for DomainU mainly include running mode interface, measurement interface, memory mapping interface, etc.

The dynamic monitoring and measurement subsystem consists of the monitoring Domain, the memory hash engine, the domain page-table manager, etc. The monitoring Domain is a special Domain which is started together with Xen and hidden inside Xen space. In the monitoring Domain, there is a VCPU which is scheduled by Xen to periodically activate the memory metric-hash engine. With the help of the Domain page-table manager, the hash engine can access any memory areas belonging to C-Domain0. According to the measurement information table, the engine calculates the hash value of the related memory area of C-Domain0 and then stores these hash values into the measurement information table.

### 4.1 The Workflow of TRIOB System

As mentioned in Section 3, The running of the TRIOB system is divided into two stages, that is, the initial trusted binding stage and the trusted running stage.

The initial binding stage is based on the trusted network connection (TNC) between the C-Domain0 and the U-Domain0[12]. In this stage, we introduce a protocol to ensure the trusted exclusive connection between the TRIOB-DomainU and remote storage resources. The protocol is listed as follows.

1) *U-Domain0 sends message to C-Domain0. The message contains image configuration, U-Domain0's public encryption key (**Kg**);*
2) *C-Domain0 stores the message into both configuration file for DomainU and the remote binding configuration table in Xen space;*
3) *According to the remote binding configuration table, Xen generates a random symmetric encryption key (**Ku**) used to encrypt I/O requests;*
4) *C-Domain0 creates and starts TRIOB-DomainU based on the configuration file;*
5) *When C-Domain0 loads the guest OS in TRIOB-DomainU, Xen checks whether the guest OS matches with the requirements based on the remote binding configuration table. For example, the version is correct or not;*
6) *The Monitoring-Domain in Xen measures the hash value (**Mu**) of the guest OS kernel in TRIOB-DomainU and stores **Mu** into the measurement information table in Xen;*
7) *TRIOB-DomainU initializes the virtual storage device, gets **Ku** from the remote binding configuration table, and gets **Mu** from the measurement information table;*
8) *TRIOB-DomainU sends message to U-Domain0. The message contains the command of binding image, EKg (**Ku**), **Mu**, image configuration, etc.;*

9) *U-Domain0 gets the message. Then it decrypts* ***Ku*** *using its private key (****Kp****), and checks whether* ***Mu*** *matches with the user's requirement;*
10) *If all above steps are passed through, U-Domain0 completes the binding process.*

It can be proved that the binding protocol is a one-to-one exclusive binding protocol. First, we ensure the one-to-one trusted connection between the two Domain0s by means of TPM/TNC specifications. Secondly, Xen is responsible for verifying the loaded guest OS kernel in TRIOB-DomainU, and generating ***Ku*** which cannot be obtained by C-Domain0. Furthermore, in the end of the binding process, user can verify the configuration of TRIOB-DomainU.

In the trusted running stage, the main task is to ensure TRIOB-DomainU to be in an absolutely close state, so that the I/O data produced by applications are securely stored into remote storage resources.

According to the users' requirements, TRIOB-DomainU can work in two modes, namely, the normal mode and the trusted mode. In the normal mode, TRIOB-DomainU is an ordinary DomainU which can be controlled by C-Domain0 as usual. In the trusted mode, C-Domain0 cannot control TRIOB-DomainU. In addition, the I/O data is attached with the metric-hash tag. In the trusted mode, C-Domain0 cannot transparently read or write the memory of TRIOB-DomainU so that TRIOB-DomainU is isolated from C-Domain0 and other DomainUs.

In the trusted mode, when the users run security-sensitive applications in TRIOB-DomainU, the processing steps of TRIOB are listed at follows:

1) *User logs on to TRIOB-DomainU by means of some remote access protocol software such as ssh, then configures the measurement policy;*
2) *The policy interpreter translates the measurement policy and registers this information into the measurement information table in Xen;*
3) *User switches TRIOB-DomainU into the trusted mode;*
4) *User starts the trusted applications;*
5) *The VCPU in monitoring Domain is periodically scheduled, then the VCPU activates the memory metric-hash engine which calculates hash of the memory areas in C-Domain0 according to the measurement information table;*
6) *The applications produce I/O operations which are received by the front-end device driver;*
7) *The metric-tag generator gets the metric information from the measurement information table to generate the metric-tag, then it sends the tag to the front-end device driver;*
8) *The front-end device driver attaches the tag onto the block I/O requests and then forwards the block I/O requests to the back-end device driver;*
9) *The front-end remote storage device driver receives these I/O requests and further transfers them to back-end remote storage device driver;*
10) *The back-end remote storage device driver receives these I/O requests. Then it separates the metric-tag and I/O data from these I/O requests, and sends them to the metric-value verifier and Linux file system, respectively;*

11) *The metric-value verifier checks the metric-tag based on the metric-value baseline table;*
12) *If the metric value matches with the metric-value baseline, the I/O data can be stored into the DomainU image; otherwise, the metric-value verifier will report an error alarm.*

In the normal mode, user can run some ordinary applications in TRIOB-DomainU. We do not elaborate the scenario in this paper.

## 5 Implementation

Currently, we have implemented the TRIOB prototype based on Xen 3.1 and Linux 2.6.18. Our implementation is described in this section.

### 5.1 Trusted Remote I/O Processing

In Xen, the front-end virtual block device driver in DomainU is "blkfront" module and the back-end virtual block device driver is "blktap" module[16]. To solve the first challenge, we firstly expand the blktap module and add two new modules (that is, remote-blkfront and remote-blkback) so that the block I/O request can be transferred to the remote storage through network. At the same time, we modify the blkfront module so that the metric-hash tag can be attached onto per block I/O request. Fig. 3 shows the detail internal structure.

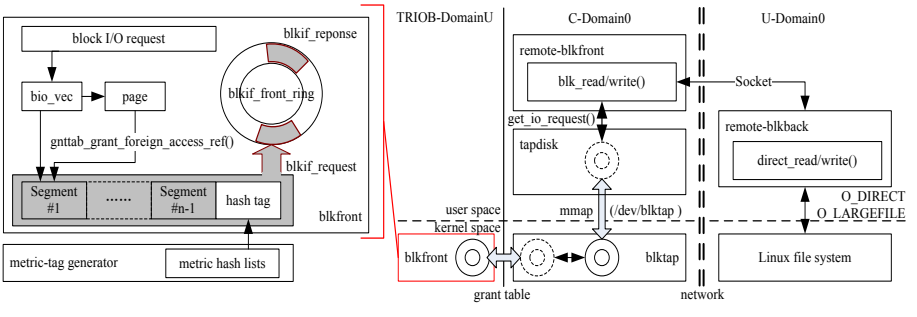


**Fig. 3.** Transfer the I/O requests attached with tag to remote storage in user's physical machine

As shown in the right part of Fig. 3, the user space part (tapdisk) of blktap maps the I/O ring memory to the user space through mmap interface. In the new remote-blkfront module, *blk_read* calls the *get_io_request* function to get the I/O requests from the user space I/O ring. In the other side, the new remote-blkback module, which has a socket connection with the remote-blkfront module, receives the I/O requests. The remote-blkback module can read or write the image with O_DIRECT and O_LARGEFILE operational mode. According to the blkif_request in I/O ring, the remote-blkback can quickly locate the block data in the image so as to ensure the I/O processing efficiency.

The I/O ring is a data structure shared between Domain0 and DomainU which contains the I/O requests. For virtual storage device, the I/O ring is called blkif_front_ring. When the blkfront module receives the block I/O request, it will interpret the request and re-organize it. In detail, it will translate the bio_vec into the blkif_request which encapsulates some segments. Amongst these segments, we select the last segment to save the metric hash value (called hash tag). The last segment is made by metric-tag generator which will get the corresponding metric hash values of some kernel memory areas in C-Domain0 through the privilege hypercall for TRIOB-DomainU.

### 5.2 The Privilege Interface for DomainU and Policy Interpreter

To solve the second challenge, firstly, we add a new hypercall which can only be called by TRIOB-DomainU. Through passing different command parameter for the hypercall, user can own some privileges as follows.

| Command | Functions |
|---|---|
| DOMU_switch_to_trusted_mode | TRIOB-DomainU in trusted mode |
| DOMU_switch_to_normal_mode | TRIOB-DomainU in normal mode |
| MONITOR_checking_register | Registering metric-related policy |
| MONITOR_checking_get_digests | Getting metric-hash value |

Secondly, we modify the xc_map_foreign_range interface. There is a lot of work based on this interface, and the most famous one is XenAccess[16]. In the TRIOB system, we modify the interface to ensure that, when TRIOB-DomainU is in the trusted mode, the DomainU can map the physical memory area in Domain0 into its virtual address space, at the same time prohibits C-Domain0 to do the same thing.

The xc_map_foreign_range interface triggers Xen's memory management functions by accessing the character driver (/proc/privcmd). We modify the ioctl function's subroutine (IOCTL PRIVCMD MMAP, IOCTL PRIVCMD MMAP-BATCH) and add the privilege checking logic. The Domain can access the interface only if it matches one of two conditions 1) the mapping Domain is the initial Domain (like Domain0), and the mapped Domain is running in the normal mode; 2) the mapping Domain is not the initial Domain (like TRIOB-DomainU), and it is running in the trusted mode.

XenAccess currently can only run in Domain0. Based on the modified xc_map foreign_range interface and with some minor patches, XenAccess can also run in DomainU. In the TRIOB system, we rename the modified XenAccess as policy interpreter (as shown in Fig. 4).

The policy interpreter gets the physical-to-machine (P2M) table of Domain0 by accessing the modified xc_map_foreign_range interface, and calculates the starting machine address of the kernel page table of Domain0 by referring to the P2M table. According to the measurement policy, the policy interpreter reads the "System.map" file to look up the corresponding starting virtual address of the measured memory areas in Domain0. Then by means of the P2M table and

the kernel page table, it can look up the corresponding page table entries (PTE) from which it can get the starting machine address (start mfn) of the measured memory areas, and then it registers these addresses into the measurement information table by calling the new hypercall with "MONITOR_checking_register" command.

### 5.3 The Memory Hash-Metric Engine, the Domain Page-Table Manager and the Monitoring Domain

The memory hash-metric engine, the Domain page-table manager and the monitoring Domain are all implemented in Xen space for two reasons. On the one hand, due to the trustworthiness of Xen, we can ensure these components are trust. On the other hand, these components need to access any memory areas belonging to C-Domain0, which is easy to do in Xen space. All of the three components cooperate together to solve the third challenge.

The memory hash-metric engine is implemented as a set of hash functions which can calculate the digest of the machine memory area. Meanwhile, we choose the MD5 hash algorithm to calculate the digest. When it is activated, the engine computes the hash of the machine memory area specified in the measurement information table and saves the hash value into the table again.

Through periodically activating the engine, the functions within the engine can be dynamically executed in Xen space. However, currently Xen does not support the concept like kernel thread in Linux OS. We found that there is an idle_domain within Xen space. The idle_domain manages all of the idle_vcpu, and these idle_vcpus and the VCPUs belonging to all other Domains are scheduled by the Xen scheduler. When the idle_vcpu is scheduled, some function codes in Xen space will run on this VCPU. Based on this idea, we introduce a new Domain (called monitoring Domain) in Xen space. The monitoring Domain has three features as follows. First, its data structure is not embodied in global domain management list, so that it is hidden in Xen space. Secondly, it shares the page table (idle_pg_table) with Xen so that it can call all of functions within Xen. Finally, it only creates one VCPU which is similar to the idle_vcpu, but the VCPU's priority is the same with the general VCPU so that it can timely activate the memory hash-metric engine.

In the measurement information table, there are some items related to machine memory regions for which the engine needs to calculate the corresponding hash value. However, the engine cannot access them directly in Xen space currently.

On the 32-bit x86 physical platform, the virtual address space of Xen in non-PAE mode covers only 64MB memory, which means that Xen can only access a maximum 64MB of machine memory for one time. To access the machine memory belonging to C-Domain0, we add a simple Domain memory mapping module (called Domain page-table manager). Now, we illustrate the way how to access the kernel machine memory of C-Domain0 as follow (Fig. 4 gives the details).

First, as mentioned in Section 5.2, we can get PTE or the starting machine address (start mfn) of the kernel memory region in C-Domain0 and the length of the kernel memory area, from the measurement information table.

Then, we allocate one-page memory from Xen heap space by calling the *alloc_xenheap_page* function. The purpose of allocating the page is to temporary use its virtual address region occupied by the page. By the *virt_to_mfn* macro, we can translate the page's starting virtual address into the corresponding starting machine page frame number (mfn). Meanwhile, by traversing the kernel page table (idle_pg_table) of Xen, we get the corresponding PTE of the page.

Finally, the Domain page-table manager temporarily replaces the PTE for Xen heap page memory with the PTE for kernel memory in C-Domain0 and refreshes TLB. Then the memory remapping work is completed. From now on the engine can directly access the kernel memory in C-Domain0. If the measured memory area is larger than one page, we can repeat the process for several times. We do not tell the way how to recover the original mapping in this paper.

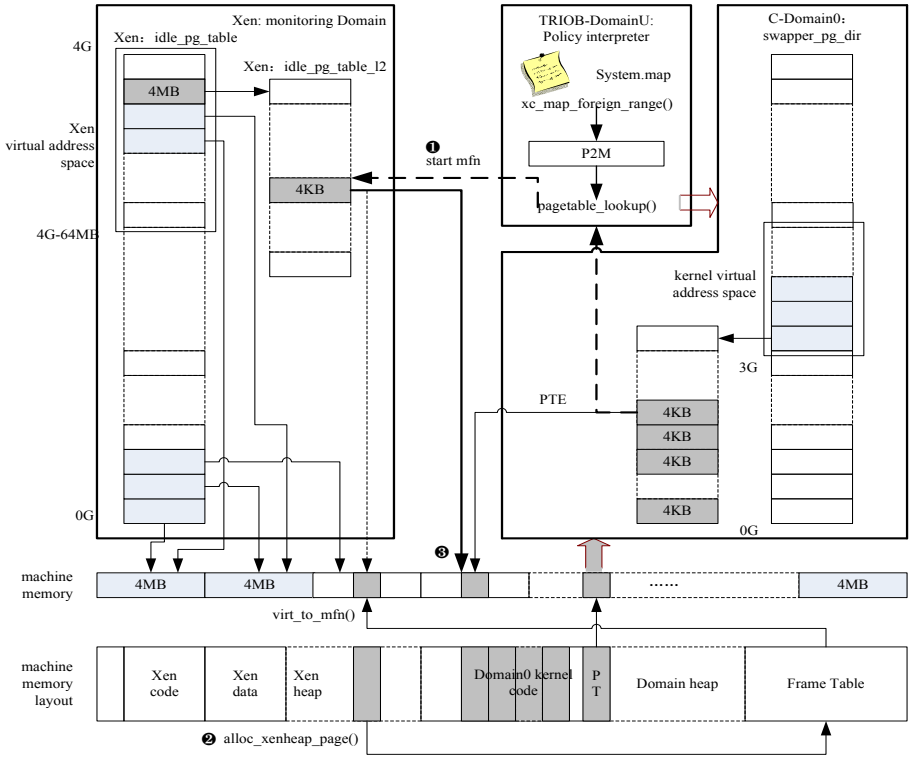


**Fig. 4.** Accessing the memory in C-Domain0 from Xen space ( ①getting the start machine page number (start mfn) and the PTE of machine memory belonging to kernel code; ②allocating the temporary machine memory from Xen heap space; ③replacing the PTE belonging to Xen heap page with the PTE belonging to kernel machine page)

## 6 Evaluation

Currently, our research group has built a virtual computing platform (called TRainbow[12]) for data center. TRainbow provides the IaaS service and TRIOB is its key component. According to the previous scenario, we get a high performance physical node from the TRainbow platform for the cloud side. Meanwhile, we also choose one lower performance node for the user side. The two machines are connected via Gigabit Ethernet network. The detail configuration of the experiment environment is shown in Table 1.

**Table 1.** Experiment environment configuration

| | | Cloud side | User side |
|---|---|---|---|
| Physical Machine | CPU | Intel Xeon E5410 2.33GHz | AMD Athlon2200+ 1.800GHz |
| | Core | 8 | 2 |
| | Cache | 6144 KB | 256 KB |
| | Memory | 16GB | 1GB |
| | Network | Gigabit Ethernet Controller | Gigabit Ethernet Controller |
| | Disk | SCSI 2*512GB | IDE 36GB |
| | VMM | Xen 3.1 (modified) | Xen 3.1 |
| Domain0 | VCPU | 1 | 1 |
| | Memory | 512MB | 512MB |
| | Guest OS | FC 6 XenoLinux 2.6.18 | FC 6 XenoLinux 2.6.18 |
| TRIOB-DomainU | VCPU | 1 | |
| | Memory | 256MB | |
| | Guest OS | XenoLinux 2.6.18(modified) | |
| | Image | 8GB | |

### 6.1 Functional Verification

The goal of functional testing experiment is to check whether the TRIOB system can detect the change of the integrity when the kernel in C-Domain0 is damaged by attacker. Currently, the system focuses on rootkits attacks against the XenoLinux in Domain0.

Our system prototype is implemented on the XenoLinux 2.6.18. During the experiment, we discovered that many of the well-known rootkits listed in Table 2, such as adore-ng 0.56, lvtes, override, phalanx-b6, cannot, without additional changes, be compiled or installed on XenoLinux 2.6.18. Besides, we cannot find other suitable rootkits. Through deeply analyzing the principle of Linux kernel-level rootkits and Xen, we conclude the major reasons as follow. First, the "/dev/mem" driver's *mmap* interface is re-implemented in Xen, as a result, traditional mem-type rootkits (like phalanx-b6) cannot easily locate the memory area belonging to the kernel. Second, Xen is the only piece of code running in ring 0, while the kernel of Domains runs in less privileged ring (ring 1 in case of x86_32). So, in XenoLinux's kernel space, traditional kernel-level rootkits cannot directly execute the privilege operations such as setting control registers. Third, most LKM-type rootkits are based on the symbols exported by the kernel, for example, the sys_call_table, as a way to hook its own code to it. But, since Linux version 2.6, some critical kernel symbols are not exported any longer and even protected in read-only memory area, which improves the system hardening process.

Therefore, we port these rootkits to the XenoLinux 2.6.18. For mem-type rootkits we resort to the C-Domain0's "System.map" to find the locations of the attacked memory area[33]. Based on this approach, we successfully install phalanx-b6. For LKM-type rootkits we need to bypass the memory protection. Traditional kernel-level rootkits can directly clear the WP bit of CR0, while XenoLinux's kernel prevents it from happening. However, there is a hypercall (HYPERVISOR update_va_mapping) by which rootkits can make the read-only memory area to be writable. Based on the hypercall, we successfully install those LKM-type rootkits (like adore-ng 0.56, lvtes, override). Meanwhile, we also implement a typical kernel-level rootkit (called hack_open) for XenoLinux 2.6.18. The rootkit is a kernel module which can modify the system call table and some other kernel memory areas. (The key function based on this hypercall is shown as follow.)

```
static int make_syscall_table_writable(unsigned long va)
{
        pte_t *pte;
        int rc = 0;
        pte = virt_to_ptep(va);
        rc = HYPERVISOR_update_va_mapping(
                (unsigned long)va, pte_mkwrite(*pte), 0);
        if (rc)
        {
                xen_l1_entry_update(pte, pte_mkwrite(*pte));
                return rc;
        }
        return rc;
}
```

Table 2 lists the public representative kernel rootkits which can be detected by the TRIOB prototype.

**Table 2.** Public representative kernel rootkits for Linux 2.6 and detection results

| rootkit name | kernel version | loading mode (type) | Can run on XenoLinux 2.6.18? | Can be detected? | integrity verification report | |
|---|---|---|---|---|---|---|
| | | | | | modify syscall_table | modify kernel text |
| adore-ng-0.56 | 2.6.16 | LKM | √ (ported) | √ | | √ |
| lvtes | 2.6.3 | LKM | √ (ported) | √ | √ | |
| mood-nt | 2.6.16 | kmem | √ | √ | √ | |
| override | 2.6.14 | LKM | √ (ported) | √ | √ | |
| phalanx-b6 | 2.6.14 | mem | √ (ported) | √ | | √ |
| suckit2priv | 2.6.x | kmem | √ | √ | √ | |
| hack_open | 2.6.18 | LKM | √ | √ | √ | √ |

### 6.2 Performance Evaluation

To further assess the validity of the system, we measure its time performance index. We deploy a few typical application workloads into TRIOB-DomainU, and test theirs total execution time by using "time" command under different I/O binding mode. The results are shown in Table 3.

**Table 3.** The execution time of different application workloads under three binding modes

| kernel-build | | | emacs | | | bzip2 | | |
|---|---|---|---|---|---|---|---|---|
| | Time(s) | % NFS | | Time(s) | % NFS | | Time(s) | % NFS |
| NFS | 456.16 | | NFS | 7.85 | | NFS | 140.34 | |
| RIOB | 371.59 | -18.54 | RIOB | 7.23 | -7.90 | RIOB | 116.64 | -16.89 |
| TRIOB | 523.57 | 14.78 | TRIOB | 8.09 | 3.06 | TRIOB | 155.24 | 10.62 |

We run three different typical application workloads under three different modes. Kernel-build is a typical I/O intensive and computation intensive application workload, and opening a large file with Emacs and then immediately closing it up is cache-sensitive, and decompressing the compressed Linux kernel source package file by bzip2 is a typical computation intensive workload.

RIOB means the DomainU (in the normal mode) bound with remote storage in the user's physical machine by our new I/O binding technology, and TRIOB represents the situation in the trusted mode. NFS means the DomainU (in the normal mode) bound with remote storage using NFS technology (version 4). As depicted in Table 3, the system's performance in the normal mode (RIOB) is better than NFS. Especially for I/O intensive workloads, the performance increases up about 19% than NFS. In the trusted mode (TRIOB) the performance lowers down about 15% than NFS under the I/O intensive workload, the overhead is significant but still in an acceptable range for users.

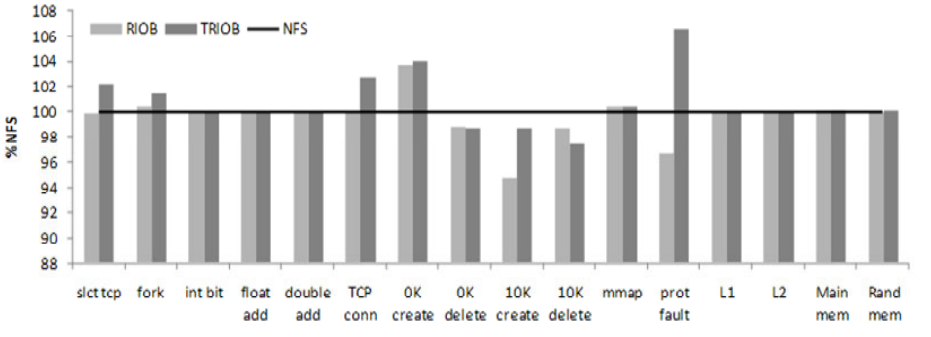


**Fig. 5.** Related lmbench results

In order to reduce the overhead, we run the **lmbench** benchmark suite in TRIOB-DomainU to help for system performance analysis. As shown in Fig.5, the overhead of TRIOB includes three parts, that is, process creation (slct tcp, fork), inter-process communication (TCP conn) and file system (file create, delete, protection fault). All of these factors are related to the "trusted I/O processing" subsystem, especially the metric-hash tag attached onto per block I/O request. In our current implementation, the tag occupied the location belonging to original block segment, which leads the times of I/O data transmission and the total transmission time to be extended. The time of accessing meta-data (related to "file create, protection fault") is critical to the overhead. For example, during compiling the Linux kernel, there will be a large number of small temporary files

to be created. The time of "0KB file create" becomes longer which results in the total compile time longer. We will deal with this problem in future work.

## 7 Related Work

VPFS[20] proposes a virtual private file system built on L4. Its implementation is divided into two parts. The front-end part provides a secure computing environment for the applications, and the back-end part reuses the traditional file system in an untrusted computing environment. As the untrusted part controls the storage resources, VPFS modifies the traditional file system with encryption functions to ensure the trustworthiness of the whole system. In contrary, in TRIOB the users can directly control the storage resources so as to avoid the additional encryption overhead. Storage Capsules[21] provides a trusted computing environment based on VM hosted on PC. The VM can run in a trusted mode (disconnecting the network) to protect user's data from attacks. Similarly, TRIOB has a trusted mode, but it focuses on the cloud computing platform. Software-Privacy Preserving Platform[22], Overshadow[23] both assume the guest OS is not trustworthy, and they both provide similar mechanism to bypass the guest OS for trust application. These works only focus on ensuring the isolation amongst the applications within the same VM, while TRIOB assumes that the guest OS in Domain0 is not trustworthy, and provides a trusted virtual machine in cloud computing platform to protect users' applications and data.

**Remote I/O Binding.** Collective[24] provides a remote I/O binding technology by which user's notebook can access the virtual storage resources in cloud computing platform. In contrary, in TRIOB a user can bind the VM in cloud computing platform with the virtual storage resources in the user's physical machine. Netchannel[25] provides a remote back-end I/O device driver for DomainU to help for the VM migration. In the TRIOB system, the back-end remote storage device driver adopts similar technology with the I/O requests verification feature. NFS, NBD, iSCSI[10][11] and other traditional remote I/O technologies are widely used in data centers. They are implemented in virtual file system layer or block device driver layer of traditional OS. Due to the splitting device driver model, there are redundant layers in the implementation when they are used in virtual computing environment. In the TRIOB system the implementation of remote I/O binding makes better of the features of the splitting device driver model to bypass some layers, and the performance is better than these traditional technologies.

**Dynamic Measurement.** To enforce the trustworthiness of the virtual computing environment, researchers have introduced a series of trust-enhancing mechanisms into data center[5][7][9]. In this field, many works are based on the TCG's TPM technology. In Terra framework[5], VMs are divided into two types, that is, gray-box and white-box. In gray-box VM, users can determine the VM's trustworthiness by the way of remote verification based on TPM. However, the TPM-based measurement can only ensure a VM's integrity at startup. To further ensure the integrity of the computing platform at running time, researchers have

bought up with many technologies related to dynamic measurement and verification of the integrity. Copilot[27] provides a dynamic memory measurement mechanism by means of a memory monitoring co-processor installed on the mainboard. In this system, the hardware can hash the memory area containing kernel text and other key components through direct memory access (DMA). In the TRIOB system, it does the similar work in Xen without adding special hardware. Pioneer[28] provides a dynamic trust root which is a running program. To verify the program's trustworthiness, pioneer periodically monitors the integrity and response time of the running program. Similarly, TRIOB dynamically monitors the integrity of Domain0 so as to determine the trustworthiness of the I/O data.

**VM Monitoring.** As virtualization technology is gradually mature, Tal Garfinkel and Mondel Rosenblum[14] propose the idea of VM introspection (VMI), an approach to intrusion detection which co-locates an IDS on the same machine as the host it is monitoring and leverages a VMM to isolate the IDS from the monitored host. In VMwatcher[18], the IDS system running in Domain0 monitors the memory in DomainU through xc_map_foreign_range interface. It can deduce the processes information in DomainU based on the data structure of task and sends this information to the anti-virus software. Lares[6] provides an active monitoring framework based on virtual machine architecture in which some hooks are inserted into the critical path in the kernel of the monitored VM. In TRIOB, we don't need to modify the kernel in the monitored VM. As for full virtual machine, by observing the related hardware behaves (like CR3 changing, TLB flush) of the process running in VM, Antfarm[17] can transparently guess the processes information. Being different from all these works, in TRIOB we propose a new function for VMI technology, that is, the monitored virtual machine can be a virtual machine (like Domain0) with higher privileges.

## 8 Conclusion

We introduced a new approach to build a trusted virtual computing environment, called TRIOB, that is geared towards data center protection and security. In TRIOB there are three key technical contributions to meet users' data security requirements. Firstly, user's data can be securely stored into theirs own storage resources through the trusted remote I/O binding technology. Second, the virtual computing environment for user is absolutely isolated from other VMs in cloud platform leveraging the trusted mode-control technique. This can ensure the integrity of user's computing environment. Finally, through the novel dynamic monitoring technique, user can timely detect the attacks against the administrator's computing environment. Dynamic monitoring can ensure the integrity of the remote I/O binding channel. We implemented a prototype system based on Xen and we quantified its security and performance properties. Our experiments show that the TRIOB system can achieve those goals above and the overhead is in an acceptable range for users.

**Acknowledgments.** This work was supported in part by the National High-Tech Research and Development Program (863) of China under grants 2009AA

01Z141 and 2009AA01Z151, the projects of National Science Foundation of China (NSFC) under grant 90718040, and the National Grand Fundamental Research Program (973) of China under grant No.2007CB310805. We would like to thank Angelos Stavrou and the anonymous reviewers for their comments.

## References

1. Barham, P., Dragovic, B., Fraser, K., et al.: Xen and the Art of Virtualization. In: Proc. of the 19th ACM Symp. on Operating Systems Principles 2003, pp. 164–177 (2003)
2. Huizenga, G.: Cloud Computing: Coming out of the fog. In: Proceedings of the Linux Symposium 2008, vol. 1, pp. 197–210 (2008)
3. Armbrust, M., Fox, A., et al.: Above the Clouds: A Berkeley View of Cloud. Technical Report No. UCB/EECS-2009-28 (2009)
4. Kaufman, L.M.: Data Security in the World of Cloud Computing. IEEE Security and Privacy 7(4), 61–64 (2009)
5. Garfinkel, T., et al.: Terra: A Virtual Machine-Based Platform for Trusted Computing. In: Pro. of the 19th ACM Symp. on Operating Systems Principles 2003, pp. 193–206 (2003)
6. Payne, B.D., Carbone, M., Sharif, M., Lee, W.: Lares: An Architecture for Secure Active Monitoring Using Virtualization. In: Proceedings of the IEEE Symposium on Security and Privacy 2008, pp. 233–247 (2008)
7. Berger, S., et al.: TVDc: managing security in the trusted virtual datacenter. ACM SIGOPS Operating Systems Review 42(1), 40–47 (2008)
8. Berger, S., Cceres, R., et al.: vTPM: Virtualizing the Trusted Platform Module. In: Proc. of the 15th Conference on USENIX Security Symposium (2006)
9. Sailer, R., Zhang, X., Jaeger, T., van Doorn, L.: Design and Implementation of a TCG-based Integrity Measurement Architecture. In: Proceedings of the 13th Conference on USENIX Security Symposium (2004)
10. Tan, T., Simmonds, R., et al.: Image Management in a Virtualized Data Center. ACM SIGMETRICS Performance Evaluation Review 36(2), 4–9 (2008)
11. Warfield, A., Hand, S., Fraser, K., Deegan, T.: Facilitating the development of soft devices. In: USENIX Annual Technical Conference 2005 (2005)
12. Sun, Y., Fang, H., Song, Y., et al.: TRainbow: a new trusted virtual machine based platform. International Journal Frontiers of Computer Science in China 4(1), 47–64 (2010)
13. Murray, D.G., Milos, G., Hand, S.: Improving Xen Security through Disaggregation. In: Proc. Of the 4th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments 2008, pp. 151–160 (2008)
14. Garfinkel, T., Rosenblum, M.: A Virtual Machine Introspection Based Architecture for Intrusion Detection. In: Proc. of the Network and Distributed Systems Security Symposium 2003, pp. 191–206 (2003)
15. Amazon Elastic Compute Cloud (Amazon EC2), `http://aws.amazon.com/ec2/`
16. Payne, B., Carbone, M., Lee, W.: Secure and Flexible Monitoring of Virtual Machines. In: Computer Security Applications Conference 2007, pp. 385–397 (2007)
17. Jones, S.T., Arpaci-Dusseau, A.C., et al.: Antfarm: Tracking Processes in a Virtual Machine Environment. In: Proc. of USENIX Annual Technical Conference 2006 (2006)

18. Jiang, X., Wang, X., Xu, D.: Stealthy malware detection through vmm-based out-of-the-box semantic view reconstruction. In: Proc. of the 14th ACM Conference on Computer and Communications Security 2007, pp. 128–138 (2007)
19. Linux kernel rootkits - protecting the system's Ring-Zero, http://www.sans.org/reading_room/whitepapers/honors/linux-kernel-rootkits-protecting-systems_1500
20. Weinhold, C., Hartig, H.: VPFS: building a virtual private file system with a small trusted computing base. In: Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008, pp. 81–93 (2008)
21. Weele, E.V., Lau, B., et al.: Protecting Confidential Data on Personal Computers with Storage Capsules. In: Proceedings of the 18th USENIX Security Symposium 2009 (2009)
22. Yang, J., Shin, K.G.: Using Hypervisor to Provide Application Data Secrecy on a Per-Page Basis. In: Pro. of the Fourth International Conference on Virtual Execution Environments 2008, pp. 71–80 (2008)
23. Chen, X., Garfinkel, T., et al.: Overshadow: a virtualization-based approach to retrofitting protection in commodity operating systems. In: Proc. of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems 2008, pp. 2–13 (2008)
24. Chandra, R., Zeldovich, N., Sapuntzakis, C., Lam, M.S.: The collective: a cache-based system management architecture. In: Proc. of the 2nd Conference on Symposium on Networked Systems Design and Implementation 2005, vol. 2, pp. 259–272 (2005)
25. Kumar, S., Schwan, K.: Netchannel: a VMM-level mechanism for continuous, transparent device access during VM migration. In: Pro. of the Fourth ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments 2008 (2008)
26. Aiken, S., Grunwald, D., Pleszkun, A.R., Willeke, J.: A performance analysis of the iSCSI protocol. IEEE MSST (2003)
27. Petroni, N.L., et al.: Copilot: a Coprocessor-based Kernel Runtime Integrity Monitor. In: Proc. of the 13th Conference on USENIX Security Symposium (2004)
28. Seshadri, A., Luk, M., Shi, E., et al.: Pioneer: Verifying Code Integrity and Enforcing Unhampered Code Execution on Legacy Systems. In: The 20th ACM Symposium on Operating Systems Principles (2005)
29. Fraser, K., et al.: Safe Hardware Access with the Xen Virtual Machine Monitor. In: Proceedings of the 1st Workshop on Operating System and Architectural Support for the on Demand IT InfraStructure, Boston, MA (2004)
30. Wojtczuk, R.: Subverting the Xen Hypervisor. Black Hat USA (2008)
31. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-3253l
32. Goodin, D.: Webhost hack wipes out data for 100,000 sites, http://www.theregister.co.uk/2009/06/08/webhost_attack
33. Lineberry, A.: Malicious Code Injection via /dev/mem. Black Hat 2009 (2009)
34. Milos, G., Murray, D.G.: Boxing clever with IOMMUs. In: Proceedings of the 1st ACM Workshop on Virtual Machine Security 2008, pp. 39–44 (2008)
35. Murray, D.G., Hand, S.: Privilege separation made easy: trusting small libraries not big processes. In: Proceedings of the 1st European Workshop on System Security 2008, pp. 40–46 (2008)
36. Dalton, C.I., Plaquin, D., et al.: Trusted virtual platforms: a key enabler for converged client devices. SIGOPS Oper. Syst. Rev. 43(1), 36–43 (2009)