2013 IEEE International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing

vMON: An Efficient Out-of-VM Process Monitor for Virtual Machines

Nan Li[2], Bo Li[12], Jianxin Li[12], Tianyu Wo[12], Jinpeng Huai[12]

[1]State Key Laboratory of Software Development Environment, Beihang University
Beijing 100191, China
{libo, lijx, woty, huaijp}@buaa.edu.cn

[2]School of Computer Science and Engineering
Beihang University
Beijing, 100191, China
linan@act.buaa.edu.cn

***Abstract*—Cloud computing service has been evolved in providing a whole virtual data center from selling scattered virtual machines (VMs). Process Monitoring of a VM is a fundamental feature to guarantee the security of the virtual data center because of the rapid growth of the malware. Existing approaches are mainly based on virtual machine introspection (VMI) technique to isolate the monitor out-of-vm and designed to inspect the VM internal processes. However, few of them consider the real time control of process execution in the VMs, such as process termination or files operation conducted by the process. Early VMI-based solutions relied on some specific OS kernel data structures, so they need to know the OS information in advance instead of identifying the OS version at runtime for operating system compatible. In this paper, we propose a novel out-of-the-box process monitor named vMON, which can not only identify different guest OS versions and reconstruct rich semantic information for the target VM processes at runtime, but also control the behaviors of processes with fine granularity. In addition, vMON provides uniform programming interfaces to support the development of application-level security tools. A prototype of vMON has been implemented in kernel-based virtual machine (KVM) hypervisor, and its effectiveness and performance have also been evaluated through several experiments. The results show that vMON can successfully identify, analyze and control the behaviors of the processes in Guest OS with acceptable performance overhead. vMon incurs 0.74%~10.20% I/O overhead and 0.003s average interface return time.**

Keywords—process control; virtualization; disk I/O; virtual machine introspection

I. INTRODUCTION

As the key enabling technology of cloud computing, virtual machines can provide users with isolated and secure computing environments, which facilitate resource sharing and greatly improves the resource efficiency of IT infrastructure. The well-known Elastic Compute Cloud (EC2) is a typical service model for providing customized, flexible and secure virtual machines. Recently, virtual data center is a trend to be another service model of cloud computing. Meanwhile, computer malwares, such as Trojans and virus, are in their limitless evolution of functionality and forms. Therefore, VM's robustness and process execution monitor become especially important.

To counteract malwares and prevent processes from being illegal manipulated and exploited, anti-malware tools have been developed. Traditional anti-malware tools are based on "in-host" monitoring mechanisms. In other words, they reside in the VM and obtain the internal activities using system programming interfaces. They rely on the integrity of host OS. And they are exposed to the malwares and could go wrong or stop working because infection. In virtualized environments, it is possible to insulate the monitor tools from the guest OS. Therefore, researchers leverage virtual machine introspection (VMI) technique to observe the VM internal activities (e.g. processes) in the virtual machine monitor (VMM) layer or in a trusted VM. However in the same time, VMI brings new problems—semantic gap and performance overhead.

The existence of semantic gap is mainly due to the reason that computer has multiple logical layers. OS layer is user-friendly, files, processes (e.g. its name and execution path) and network connections can be directly observed in this layer. While in hardware or VMM layer, only binary codes can be obtained. In order to bridging the gap, many recent works come up with several approaches. Some of them overcome the challenge and complement it for the special operating system by using the guest OS kernel data structure or just assume that guest OS version is the same with host OS version. Some of them focus on generating introspection tools automatically [6, 7, 8]. There are also some introspection-based tools aiming at controlling system calls or files access from outside of VMs. To meet the needs of virtual data center complex circumstances, several methods [14, 15, 16] have been proposed to provide OS-compatible monitoring. While they cannot provide fine-grained control of VMs running processes which are very necessary and important to security application like anti-malware tools.

Whenever we discover a suspect process running in the target VM, it's necessary to block the process's I/O operations to prevent further damages or loss; when we confirm that it's a malicious process, it should be terminated immediately. For users, it would be a friendly way to allow them to configure their own sensitive files that cannot be modified by the suspect process.

In order to realize the whole out-of-vm monitoring and control procedure automatically described above, we introduce vMON, a fine-grained monitor for monitoring and controlling processes in the VMs with different operating systems. vMON is transparent to the target guest OS through VMI techniques. In order to support different OS versions, vMON combines the OS kernel code analysis and hardware virtualization to bridge the semantic gap and rebuild the abundant process related information of the VM. Above all, vMON provides several significant functions to control the VM's process execution from outside of the target guest VM which we call it out-of-

978-0-7695-5088-6/13 $31.00 © 2013 IEEE
DOI 10.1109/HPCC.and.EUC.2013.194

vm method. It is meaningful for security tools to stop the malicious process running or prevent the operation of the suspect process on the disk in time. In summary, vMON makes the following contributions:

- We propose a novel technique to identify different guest operating systems at runtime and reconstruct the information based on kernel data structure offsets collections (KDS-OC, differ in kernel versions). Unlike existing approach [15] which requires installing a module in the guest OS, vMON doesn't need install or run any codes in the guest OS. At any time of the VM runtime, vMON can identify the corresponding KDS-OC automatically.
- We undertake a limited but meaningful attempt to control the VM internal running processes from outside of the guest OS, such as killing a specified process and limiting its access to virtual disks. To prevent a suspect process from modifying users' sensitive files, we present File-to-Block Mapper which bridge the semantic gap between OS file-level and disk block-level on images with raw format.
- We provide a user-friendly application programming interface (vMON APIs) based on KVM, which can be used by programmers to develop various real-time process monitoring tools.
- We have implemented the above techniques into vMON, and prove its availability through comprehensive experiments. Simultaneously, we have applied vMON APIs to a user demo which is used for managing VMs in the same physical machine and control VMs' internal running processes, the evaluation results show vMON can monitor and control the behaviors of processes in a real-time manner.

The rest of this paper is structured as follows: In Section II, we present the related work in virtual machine introspection for different purposes. Next, Section III states the problems to be resolved by vMON and presents system overview. The implementation details of vMon are described in Section IV. Then we evaluate the effectiveness and performance in Section V. Finally in Section VI we make a conclusion and raise the future direction of vMON.

II. Related Work

VMI is proposed by Garfinkel and Rosenblum in Livewire [3], and it has been widely used in addressing the security problems of computer systems. Because VMM resides beneath the guest OS, all resources of the VM can be monitored and controlled from the VMM layer. However, the security applications based on virtualization face the challenge of semantic gap. To address this challenge, many research works [6, 7, 8, 9] focus on automatically generating VMI tools. Virtuoso [8] is a novel system which enables automatic VMI tool generation. POG (Process Out-Grafting) [6] bridges the semantic gap by using a trusted VM to monitor the target VM's suspect process and supports existing user-mode process monitoring tools. VMST [7] is similar to POG, but it aims to inspect the whole OS and monitors all the instructions in the target VM. VMST and POG both need another secure VM to help monitor the untrusted VM. So they will increase resource usage and leads to resource waste.

LibVMI [15] is a virtual machine introspection library based on the related XenAccess [16] library supporting Xen and KVM. It provides APIs for accessing the VM's memory and supports a variety of different OSes by installing a module in the VM in advance and writing a configuration file for each VM. LibVMI also presents a page cache algorithm to improve the performance. VMDriver [14] is proposed to achieve good compatibility. It separates the event interception point and guest OS semantic reconstruction in VMM level and in management domain respectively, so that it can rebuild the OS view based on different 'OS driver'. CloudSec [13] is a monitoring appliance used for inspecting VM's memory in the IaaS (Infrastructure-as-a-Service) cloud platform. These systems only present procedures of VM internal system call or processes list rebuilding without any control methods.

From another perspective, Filesafe [4] uses VMI to protect user's sensitive files from outside of the VM. It bridges the file-level and block-level semantic gap by designing a FSP (File System Parser) which is related to the file system type rather than operating system version. Filesafe limits file operations without differentiating processes. It will block all I/O requests to the sensitive files no matter whether they come from a suspect process or not.

vMON is proposed in this work. It combines observation and control functionality to monitor the VMs' internal running processes using an out-of-vm method. At the same time, it puts much emphasis on the compatibility of multiple Linux operating systems and file systems. As a basic security tool, vMON also exposes API to programmers and other applications which can be used to build their own security tools and applications.

III. System Overview

A. Problem Statement

Based on the above analysis of related work, more and more researches focus on the security of VMs internal processes execution. However, in consideration of the virtual data center scenario, there are several new challenges described as following:

- Many existing introspection tools are mainly appropriate for the specific version operation system, while the virtual data center may provide users guest VMs with different kinds of operation systems. We need to distinguish these versions without installing any code or running any process in the guest OS online.

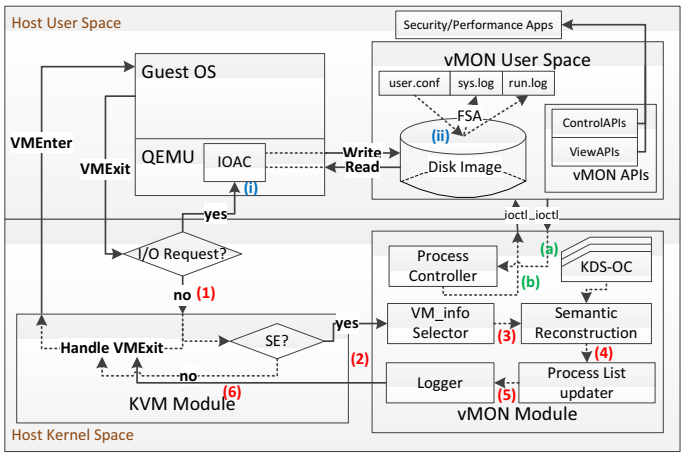

Fig.1. vMON Architecture

- Most out-of-vm monitor tools only show the guest OS internal processes information. However, it is necessary to stop a given process which is confirmed malicious from out of the VM in time. So controlling the processes from the outside of VMs is meaningful for anti-malware tools of the virtual data center.
- Data files in the virtual data center are important to users. In particularly, some of them shouldn't be accessed by suspect process (We call it process-sensitive files in this paper). Few works focus on preventing the suspect process from operating user process-sensitive files online through disk I/O control in the virtualization environment.

B. *Scope and Assumptions*

Although vMON aims at automatically monitoring and controlling various versions of operation systems, there are some limitations to our techniques. First, vMON is, by design, able to identify the guest OS kernel related information based on its kernel symbol's address and reconstruct semantic information by kernel data structure offset collection, so the guest OS kernel data structure related knowledge must be available. Secondly, vMON has to analyze the guest OS image offline because different file systems organize and manage files in different ways. As a proof-of-concept, we focus on Linux OS, on top of the widely used x86 architecture. Currently, vMON supports multiple versions of guest operation systems, such as Ubuntu, CentOS, Debian (Lenny/squeeze/wheezy) and Fedora. For the protection of process-sensitive files, we support Ext2, Ext3 and Ext4 file system. However, with our proposed techniques, it can be extended to other operation systems and file systems (e.g. Windows and NTFS) with effortless ease.

Meanwhile, we assume that our own VMM (e.g. QEMU/KVM) is trusted since some existing researches ensure this and VMM's source code is more reliable than the OS kernels. In addition, we allow the attackers to have root privilege of the guest virtual machines, and they can take control of the guest VM's operating system kernel. vMON should work correctly even when the monitored VM has been compromised. We also assume that in the virtual data center, all VMs run the supported OSes by vMON. The target VM will not be stopped monitoring even when it is migrated from one host machine to another. Finally, vMON is designed to work with hardware virtualization extensions available in commodity CPUs (Intel VT-x or AMD-V).

C. *System Overview*

Figure 1 shows the architecture of vMON with three primary components. Before presenting each component in detail, we point out the preparation work that our system should do. First, if users want to enable process-sensitive files control, they must modify the VM's disk image profile (Each disk image has a corresponding profile.) to define the process-sensitive files and their permissions (`READ-ONLY` or `NO-RW`). Then vMON module must get the `KDS-OC` (Kernel Data Structure Offsets Collection) ready to support multiple OS versions of guest VMs. Whenever the VM is powered on with our modified QEMU, vMON achieves the user's profile to create the sensitive disk blocks array. Once vMON discovers that the guest VM has finished initialized the `IDT` (Interrupt Descriptor Table) entry address, vMON decides which `KDS-OC` should be used by this guest VM. And afterwards the guest VM can be monitored at the process granularity by vMON in real time.

1) Process Analyzer component is located in KVM module and vMON module. It's the key portion of vMON which supports multiple operating systems at runtime. When VMExit happens and is not an I/O request, the `SE` (Sensitive Event) will charge whether it's an event we are interested in, such as setting `CR3` value. Then vMON can intercept the event, utilizes the `VM_info` which was created when the VM started or was migrated to the host machine and `KDS-OC` to rebuild the dynamic VM-internal information view (3) (4). Finally vMON updates the view list based on update-pNode algorithm (5), records the log and goes back to KVM module (6).

2) Process I/O sensor consists of IOAC (Input /Output Authority Check) online and FSA (File System Analyzer) offline which reside in the user space of the Host OS. It is

based on both users profile and file-blocks mapping algorithm (details in Section IV).

3) Process Controller enables the virtual data center provider to send control command, such as killing a specific process in the monitored VM (a). According to the command, VM Controller conducts corresponding modification on the monitored running VM's related kernel data value to achieve the equal effect with the command executed in the VM.

vMON also provides a set of APIs to the out-of-vm security or performance applications, including control APIs and view APIs. These Applications can utilize these APIs to monitor or control the target VM internal processes without knowledge of VMM and guest OS kernel data structures.

IV. IMPLEMENTATION

We have implemented vMON as a loadable kernel module in cooperation with the modified open source hypervisor KVM (version 2.6.32.27) and QEMU (version 0.12.5). As shown in Figure 1, our implementation only adds about 50 SLOC in KVM and 565 SLOC in QEMU. vMON module and user space program consist of 1506 and 314 lines of C code, respectively. In the rest of this section, we will discuss vMON implementation with Intel VT support in details.

A. Process Analyzer for Multiple Operating Systems

Inspired by [5] mentioned guest OS identification we attempt to find factors concerned with guest OS kernel version. After conducting a large number of experiments, we focus on `IDT` (Interrupt Descriptor Table) entry address and `SCT` (system call table, for Linux) entry address.

TABLE I RELATIONSHIP BETWEEN DISK/MEMORY AND IDT/SCT EA

Linux Distribution	Kernel Version	Disk(GB)/ Mem(MB)	IDT EA/ SCT EA
ubuntu-10.04.4	2.6.32-38	*5/512*	0xc0767000/0xc01033a0
ubuntu-10.04.4	2.6.32-38	*10/512*	0xc0767000/0xc01033a0
ubuntu-10.04.4	2.6.32-38	*5/384*	0xc0767000/0xc01033a0

From Table I, we confirm that IDT and SCT entry addresses of guest OS have nothing to do with disk and memory sizes. Next step, we look into several distributions of Debian, CentOS and Fedora, the result is to be expected shown in Table II.

Notice that for the same kernel version 2.6.32-5-486 (row 5 and 8 in Table II) the `IDT EA` is not the same, but the `STC EA` is identical. Based on the above inductive experiment, we made a conclusion, Linux VMs always have different system call entry addresses even their operating systems kernel versions only have subtle differences. So vMON builds kernel data structure offsets collection (`KDS-OC`) for each VM according to their system call entry addresses.

As mentioned in section II, many related works have proposed to get VM's internal running processes from outside of the VM based on process switch interception. For Linux, context switches are one of the key techniques to allow multiple processes to share a single CPU. Intel VT, a hardware virtualization technique, introduces a new CPU execution mode named root mode that allows the hypervisor to run below ring 0. A guest OS runs in the non-root mode so that privileged OS instructions will cause VMExit. For example, whenever process context switch happens in the VM, the state of guest OS is stored in the `VMCS` (virtual machine control structure) maintained by the hypervisor and the VM exits to KVM. Then KVM analyzes the exit reason. If it is an I/O request KVM will submit it to QMEU.

TABLE II RELATIONSHIP BETWEEN KERNEL VERSION AND IDT/SCT EA

Linux Distribution	Kernel Version	IDT EA/ SCT EA
debian lenny	2.6.26-2-28	0xc036c000/0xc010388c
debian-6.0.7	2.6.32-5-686	0xc135f000/0xc1003190
debian-6.0.7-standard	2.6.32-5-686	0xc135f000/0xc1003190
debian-6.0.7-standard	***2.6.32-5-486***	***0xc132f000/0xc10030e0***
debian-6.0.6-rescue	2.6.32-5-686	0xc135f000/0xc1003190
debian-6.0.6-standard	2.6.32-5-686	0xc135f000/0xc1003190
debian-6.0.6-standard	***2.6.32-5-486***	***0xc132d000/0xc10030e0***
debian-6.0.6-desktop	2.6.32-5-686	0xc135f000/0xc1003190
debian-7.0.0	3.2.0-4-686-pae	0xc13de000/0xc12c26c8
debian-7.0.0-standard	3.2.0-4-686-pae	0xc13de000/0xc12c26c8
debian-7.0.0-standard	***3.2.0-4-486***	***0xc138f000/0xc1282c80***
debian-7.0.0-rescue	3.2.0-4-686-pae	0xc13de000/0xc12c26c8
debian-7.0.0-rescue	***3.2.0-4-486***	***0xc138f000/0xc1282c80***
debian-7.0.0-desktop	3.2.0-4-686-pae	0xc13de000/0xc12c26c8
debian-7.0.0-desktop	***3.2.0-4-486***	***0xc138f000/0xc1282c80***
Fedora-17	***3.3.4-5.fc17.i686.FAE***	***0xc0b64000/0xc0943c98***
Fedora-18	3.9.4-200.fc18.i686.FAE	0xc0bf0000/0xc09a98c8
Fedora-18-Desktop	3.9.4-200.fc18.i686.FAE	0xc0bf0000/0xc09a98c8
CentOS-5.9	***2.6.18-348.e15***	***0xc0704000/0xc0404f04***
CentOS-6.4	2.6.32-358.e16.i686	0xc0a57000/0xc084a2a8

While in this case, the reason must be `handle_cr`. Before setting the next-process's `PGD` (Page Global Directory) to `CR3`, KVM notifies vMON to reconstruct the current process (before switch). In this procedure, vMON makes use of `ESP` register to find out `GVAs` (Guest Virtual Address) of the `thread_info` structure and `task_struct` structure. Then `KDS-OC` helps vMON find out locations of the process basic information, such as `pid` and `name`. Figure 2 presents work flow of the whole procedure. We can see that no matter the guest OS is powered on or is migrated from other physical machine, vMON is able to identify its kernel version and select the corresponding `KDS-OC`.

Although we only implemented the reconstruction of process view, without loss of generality, vMON can retrieve more information from outside of guest OS by analyzing OS kernel data structure and extending the `KDS-OC`.

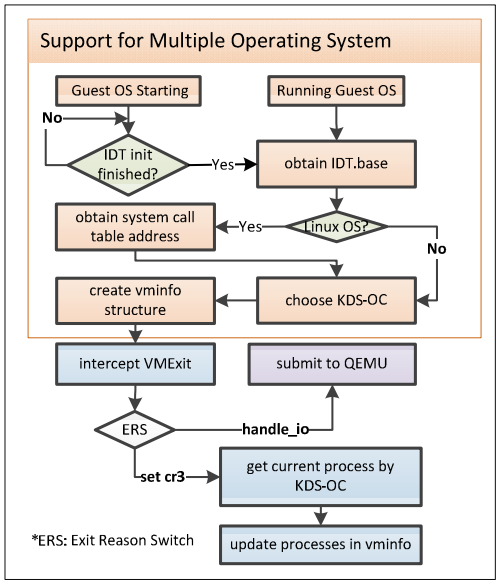

Fig. 2. Workflow of process analyzer

B. *Process Control from Outside of the VM*

Most researches of the existing process monitor tools for cloud computing or virtual data center just provide process information. Some of them support VM-level control but rarely provide finer granularity control for VM, such as process-level control. In order to control the VM internal processes from outside of guest OS, the key idea used by vMON is to *do the least and gain the most*. It means that vMON takes full advantage of operating system inherent mechanism to hit our target. We begin with attempting to kill a user-appointed process in his Linux VM.

As far as we know, in Linux OS, `kill -9 pid` is a useful command to terminate the process whose `ID` is `pid`. Actually, the principle of kill command is to send a non-inhibitable signal (`SIGKILL` with value 9) to the process. Illuminated by this OS mechanism, vMON simulates the procedure of sending `SIGKILL` to the user-appointed process. Process control block is stored in a data structure called `task_struct`. When its process X's turn to run on the CPU, firstly, the `pending` field of its `task_struct` will be checked to make sure there are no unhandled signals. So if vMON modifies the `pending` filed and related fields to 'send' `SIGKILL` successfully, the rest work will be passed on to the guest OS which helps vMON to kill process X.

QEMU (Quick EMUlator) provides a monitor console for interacting with QEMU and a user-friendly interface to interact with KVM module based on `ioctl`. In Figure 3, we extend the monitor interface to define commands `kill` and `reboot` by following QEMU monitor protocol (QMP). Command `kill` accepts a valid `process ID` in the running VM. Once it is issued, QEMU notifies vMON's process control module to set `pending`, `flags` and `preempt_count` of `thread_info` data structure. Command `reboot` is able to reboot the VM by modifying the value of `pgd`.

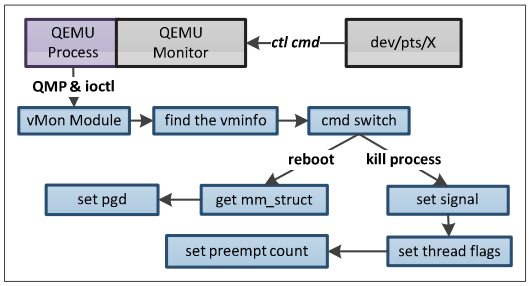

Fig. 3. Workflow of process control

C. *Process I/O Control*

As mentioned in Section III, we aim at controlling the suspect process operations on the sensitive files via intercepting VM disk I/O requests. This produces a gap between the file-level from inside of the guest OS and the block-level from the hypervisor (out of the guest OS). It is another kind of semantic gap which should be filled in using introspection. For example, users only care about the files or directories in the VM. They point out which files in the guest OS instead of which blocks in the disk are sensitive to read or write. We design File-to-Block Mapper to bridge the semantic gap between the two different levels.

Before describing the details about the Mapper, we introduce `Data Manager` and `IOAC` (I/O Authorization Checker) module. `Data Manager` takes charge of managing related files of process-sensitive files control, including `disk.info`, `user.conf`, `run.log` and `sys.log`. `IOAC` consists of `I/O Interceptor` and `Authorization Checker`. They are used for inspecting I/O request, including which process (`pid`) issues it, and reacting to the request (deny or allow). We implement `IOAC` by analyzing QEMU source code related with I/O handle procedure.

File-to-Block Mapper is a core component of FSA. It aims to map the files those users configure as `READ-ONLY` or `NO-RW` to the blocks in the VM image disk. The Mapper reads user's profile to get the names of process-sensitive file and transforms them to blocks numbers. File system is one of the most important portions of operating system. It manages files' allocation and organization. Ext3 (third extended file system) is a journal file system commonly used by Linux kernel. It is the default file system for many popular Linux distributions, including Debian Squeeze. We take it as an example: Ext3 divides the disk into several `block groups` (`bg`), and each block group consists of `super block`, `group descriptor table` (`gdt`), `block bitmap`, `inode bitmap`, `inode table` and `data blocks`. File-to-Block Mapper analyzes the `super block` (`1024 KB`) to acquire the basic information of the file system, such as block size, blocks per group, inode size, etc. As each file is identified by an inode, the problem is converted from file-to-block to inode-to-block. By tracing Ext3 internal data structure, we obtain the

inode address formula shown in Table III. And then we can obtain the blocks those sensitive files occupy. Specially, Algorithm 1 shows this procedure of Ext3.

TABLE III FORMULAS OF CALCULATING INODE ADDRESS

bg_num=(*inode* -1)/*inodes_per_group*
inode_offset=(*inode* -1)%*inodes_per_group*
inode_addr=*gdt*[*bg_num*](9~12*bytes*)**block_size*+*inode_offset***inode_size*

Algorithm 1: File A blocks location

1: **Require**: **File A**'s inode address (*addr*) ready; file system type *x* is known; **Load** (*i, j*) is a function to read data from the disk image from address *i* to address *j* in bytes; *BlocksArray* stores the blocks belong to File A. It extends by using its **add** function; **IndirectAnalysis, DoubleIndirectAnalysis** and **TripleIndirectAnanlysis** are iteration functions to get all three levels' blocks of **File A.**
2: **FileBlocksLocation**(*addr*, *x*):
3: **switch**(*x*):
4: **case** *EXT3*:
5: *BlocksArray*.**add**(**Load**(*addr*+41,*addr*+88));
6: if **Load**(*addr*+89,addr+92) != 0:
7: *BlocksArray*.**add**(**IndirectAnalysis**);
8: if **Load**(*addr*+93,*addr*+96) != 0:
9: BlocksArray.**add**(**DoubleIndirectAnalysis**);
10: if **Load**(*addr*+97,*addr*+100) !=0:
11: *BlocksArray*.**add**(**TripleIndirectAnanlysis**);
12: break;
13: **case** *OTHERS*:
14: **…**
15: **return** *BlocksArray*;

Although we implement File-Block Mapper for Ext3, we find it works well in Ext2 file system without any modification. Moreover, it needs little change in Algorithm 1 to support Ext4. In addition, our Mapper can be used into a disk offline-scanning tool by iteration of finding all files stored in the disk.

V. EXPERIMENTS AND EVALUATION

In this section, we conduct several experiments to evaluate effectiveness and performance of vMON. To evaluate the effectiveness of vMON, first we look at diversity of operating systems to prove they are compatible. Next we discuss whether process control from outside of the VM takes effect. Finally, we examine the process-sensitive files operation control component whether works effectively. To illustrate the performance overhead results, we use several standard benchmarks.

The experiments were conducted on physical machine of Intel(R) Core(TM) i7 CPU with 4G memory. Our host OS is Debian 6 with Linux kernel 2.6.32-5-amd64. Virtual machines will be used in the experiments are listed in Table IV.

A. Effectiveness

1) Support for Multiple Operating Systems

vMON is designed to monitor processes in the different guest operating systems out of the VMs without installing or running any code in the guest OS. We start all the VMs in Table IV with vMON, and using a demo implemented through vMON APIs to list all the VMs running in the host OS (with the option `-m`) and get their all internal running processes respectively (with the option `-P uuid`), we present the latest three processes of them in Figure 4.

```
root@debian6:/home/linan/userDemo# ./printCurrentDetails -P\
> 4a29dbc2-4f2f-11de-9b6d-00188b1b8d0d
        \____proc_info: [pid: 1185      ] [cmd: exim4]
        \____proc_info: [pid: 1178      ] [cmd: getty]
        \____proc_info: [pid: 1177      ] [cmd: getty]
root@debian6:/home/linan/userDemo# ./printCurrentDetails  -P\
> 3a29dbc2-4f2f-11de-9b6d-00188b1b8d0d
        \____proc_info: [pid: 8867      ] [cmd: sleep]
        \____proc_info: [pid: 1468      ] [cmd: getty]
        \____proc_info: [pid: 1466      ] [cmd: getty]
root@debian6:/home/linan/userDemo# ./printCurrentDetails -P\
> 5a29dbc2-4f2f-11de-9b6d-00188b1b8d0d
        \____proc_info: [pid: 3771      ] [cmd: getty]
        \____proc_info: [pid: 3770      ] [cmd: getty]
        \____proc_info: [pid: 3769      ] [cmd: getty]
root@debian6:/home/linan/userDemo# ./printCurrentDetails -P\
> 6a29dbc2-4f2f-11de-9b6d-00188b1b8d0d
        \____proc_info: [pid: 821       ] [cmd: kcryptd]
        \____proc_info: [pid: 820       ] [cmd: kcryptd_io]
        \____proc_info: [pid: 787       ] [cmd: kdmflush]
root@debian6:/home/linan/userDemo# ./printCurrentDetails -P\
> 7a29dbc2-4f2f-11de-9b6d-00188b1b8d0d
        \____proc_info: [pid: 1410      ] [cmd: kworker/0:1]
        \____proc_info: [pid: 1409      ] [cmd: kworker/0:0]
        \____proc_info: [pid: 1408      ] [cmd: kworker/0:2]
root@debian6:/home/linan/userDemo# ./printCurrentDetails -P\
> 8a29dbc2-4f2f-11de-9b6d-00188b1b8d0d
        \____proc_info: [pid: 0 ] [cmd: Idle]
        \____proc_info: [pid: 1132      ] [cmd: wuauclt.exe]
        \____proc_info: [pid: 1920      ] [cmd: alg.exe]
        \____proc_info: [pid: 1888      ] [cmd: ctfmon.exe]
root@debian6:/home/linan/userDemo# ./printCurrentDetails -P\
> 9a29dbc2-4f2f-11de-9b6d-00188b1b8d0d
        \____proc_info: [pid: 1800      ] [cmd: wuauclt.exe]
        \____proc_info: [pid: 1796      ] [cmd: svchost.exe]
        \____proc_info: [pid: 672       ] [cmd: VSSVC.exe]
```

Fig. 4. Using vMON to show VMs internal running proces

2) Process Control

One of vMON features is allowing administrator (of the virtual data center) to terminate the confirmed malicious process of the VM immediately from outside of the VM. In order to present the scenario of killing a specific VM internal process out of the VM intuitively, vMON started VM1 with `-monitor stdio` option so that we can send commands and observe the result directly. Firstly, we run `run_process` in the back end with `&` in VM1 and we get its `pid` is `1272`. Then we use extended command `kill 1272` on QEMU monitor to terminate `run_process`. Finally we use `ps -ef | grep 1272` to verify that the process is killed successfully. Figure 6 shows the screenshots of the whole procedure (The top one is in the guest OS and the bottom one is QEMU Monitor in the host OS).

TABLE IV. VIRTUAL MACHINE CONFIGURATION IN EXPERIMENT

VM NO.	OS Distribution	Kernel Version	VM UUID	Disk(GB)/Mem(MB)	File System
VM0	Debian 5.0	2.6.26-2-28	3a29dbc2-4f2f-11de-9b6d-00188b1b8d0d	1.0/512	Ext2
VM1	Debian 6.0.7	2.6.32-5-686	4a29dbc2-4f2f-11de-9b6d-00188b1b8d0d	2.0/512	Ext3
VM2	Debian 7.0.0	3.2.0-4-686-pae	5a29dbc2-4f2f-11de-9b6d-00188b1b8d0d	5.0/512	Ext4
VM3	Ubuntu 10.04.04	2.6.32-38-generic	6a29dbc2-4f2f-11de-9b6d-00188b1b8d0d	10.0/512	Ext4
VM4	Ubuntu 12.04.02	3.5.0-23-generic	7a29dbc2-4f2f-11de-9b6d-00188b1b8d0d	10.0/512	Ext4

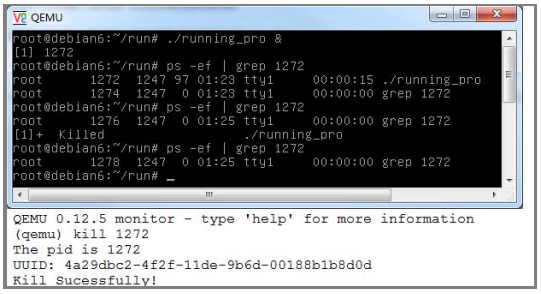

Fig. 5. Terminate a specific process from out of the VM

3) Process-sensitive Files Accesss Control

The other important feature of vMON is controlling process-sensitive I/O. We create files named `f-test-0` and `f-test-1` in VM0 and VM1. Assume that `run_malware` is a suspect process and `run_normal` is a trust process. They will read/write `f-test-0`/`f-test-1`. Then we modified these VMs user profiles to make `f-test-0` `READ-ONLY` and `f-test-1` `NO-RW`, respectively.

From the result we can see that `run_normal` is able to read and write these above two files; however `run_malware` can only read `f-test-0` and cannot do anything to `f-test-1`. At the same time, related logs have been recorded in `run.log` of VM1.

B. Performance

From the system implementation in Section IV, we notice that the major overhead of vMON comes from `IOAC` in QEMU, the online process analyzer in KVM module and vMON module. `IOAC` component is used for checking the block (to be read or written) is whether in the process-sensitive blocks array. Therefore, we anticipate that it tends to have larger overhead if sensitive files occupy more blocks. For the process analyzer, the number of processes may affect the overhead.

We use UnixBench (Version 5.1.3), which is to provide a basic indicator of the performance of a Unix-like system, to evaluate the VM overall performance. We choose VM1 in this experiment and create two files (`40M-0`, `40M-1`) in it. Their sizes are both `40MB`. So there are 20000 process-sensitive blocks. The number `0` and `1` in the file's name represent the file's I/O authority is `READ-ONLY` and `NO-RW` respectively. The file system of VM1 is Ext3 with 4096KB block size. The results are shown in Figure 7. Compared with the original system, VM in vMON overall performance loss is about 16.23% and the major loss comes from system call overhead (5). However some indexes such as `File Copy` ((1) ~ (3) in Figure 7) which is relative to I/O performance overhead is relatively small. Moreover, there is almost no performance loss in other indexes ((4), (6), (7)).

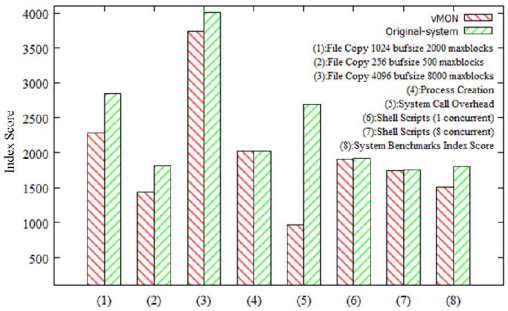

Fig. 6. VM overall performance with UnixBench

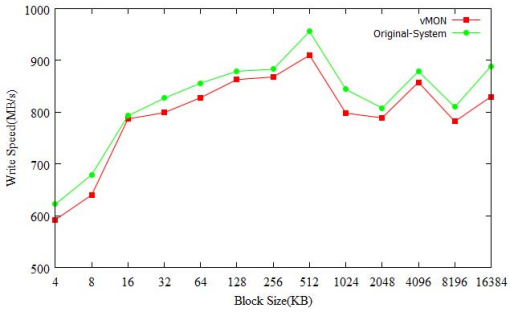

Fig. 7. Write a 16MB file speed comparison

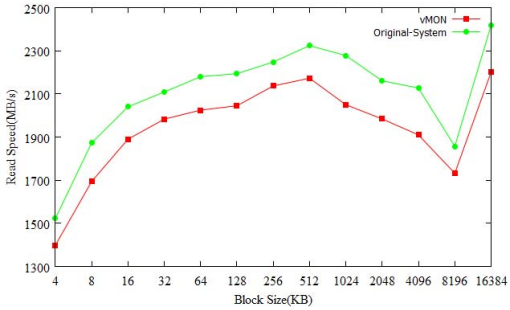

Fig. 8. Read a 16MB file speed comparison

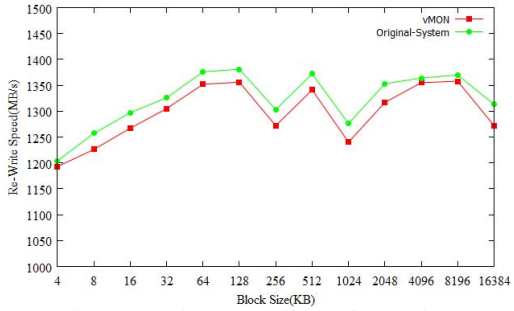

Fig. 9. Re-Write a 16MB file speed comparison

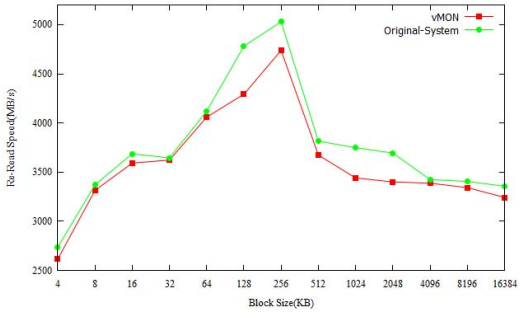

Fig. 10. Re-Read a 16MB file speed comparison

From the above analysis, we measured the read/re-read and write/re-write speeds with iozone using the same environment of UnixBench experiment. By running iozone in the VM started by vMON and original QEMU-KVM 50 times respectively, we calculate the VM's average speeds of reading/writing and re-reading/re-writing a 16MB file with different blocks sizes (4KB~16384KB). As described from Figure 8 to Figure 11. With the blocks size increasing, vMON and the original QEMU-KVM's I/O speeds both climb up and then decline. And the average slowdown by vMON is acceptable range from 0.74% and 10.20%.

Particularly, we test the cost of process analyzer with different numbers of the running processes. In VM0, there are about 34~40 processes, and vMON consumes about 1.3μs for each interception analysis. There are about 55~ 60 processes in VM1 and the cost of each interception analysis are also about 2.2μs. Moreover, we test the costs of `kill` command used by in-guest application and our out-of-vm method. The average costs are 1.072s and 1.037s, respectively. It shows that our control to process is more efficient. For the offline FSA (File System Analyzer), we use it to scan a disk image with common configuration. It only takes 5.320865s to find out all 29911 files and 2637 directories in the disk image.

Finally, we write a demo with vMON APIs to get the VM's running processes. By running the demo 500 times, the average time it consumes is 0.002s. The average elapsed time is so small that it proves vMON enable real-time monitoring.

VI. CONCLUSION

In this paper, we have presented vMON, an out-of-vm system for observing and controlling processes in the VM based on VMI technique. It is implemented to support multiple Linux operating systems and file systems. vMON is transparent to the VMs with no need to install or run any codes in the guest OS. For the control of specific process, vMON provides process manipulation mechanisms which can terminate a process from outside of the guest OS and can also protect process-sensitive files from being illegally accessed. Moreover, we design and implement vMON APIs, a set of user-friendly and efficient programming interfaces, which facilitate the development and programming of security application developers work more convenient. From the performance results shown above, vMON is an effective external process monitor and provides real time monitoring APIs. It is suitable for the complex virtual data center circumstance.

We are currently extending vMON to support more guest operating systems, including Windows/Unix Series. At the same time, although I/O performance is closed to the original system, there is still room for improving vMON in VMExit interception to reduce system call overhead. We will add a cache mechanism in vMON to avoid repeatedly intercepting the same events so as to reduce the processing time of the process analyzer. We will also extend vMon to support more file systems and image formats (e.g. `qcow2`).

ACKNOWLEDGMENT

The work in this paper has been supported in part by the China 863 program (No. 2011AA01A202), National Basic Research Program of China (973) (No. 2011CB302602), National Nature Science Foundation of China (No. 61202424, 61272165, 91118008), New Century Excellent Talents in University 2010 Beijing New-Star R&D Program under Grant No. 2010B010 and SKLSDE-2012ZX-21.

REFERENCES

[1] Kivity, Y. Kamay and D. Laor "KVM: the Linux Virtual Machine Monitor," in Proc. Linux Symposium vol. 1, pp. 225-230, 2007.

[2] F. Bellard. "QEMU, a Fast and Portable Dynamic Translator," in ATEC, pages 41–41, 2005.

[3] T. Garfinkel and M. Rosenblum. "A virtual machine introspection based architecture for intrusion detection,". in Network and Distributed System Security Symposium, 2003.

[4] Junqing Wang, Miao Yu, Bingyu Li, Zhengwei Qi and Haibing Guan, "Hypervisor-based Protection of Sensitive Files in a Compromised System," SAC'12, March 25-29, 2012.

[5] Christodorescu, M., Sailer, R., Schales, D. L., Sgandurra, D., and Zamboni, D. "Cloud Security Is Not (Just) Virtualization Security," in CCSW '09:Proceedings of the 2009 ACM workshop on Cloud computing security (New York, NY, USA, 2009), ACM,pp. 97–102.

[6] D. Srinivasan, Z. Wang, X. Jiang, and D. Xu, "Process out-grafting: an efficient "out-of-vm" approach for fine-grained process execution monitoring," in Proc. of the 18th ACM conference on Computer and communications security (CCS'11), October 2011.

[7] Y. Fu and Z. Lin, "Space Traveling across VM: Automatically bridging the semantic gap in virtual machine introspection via online kernel data redirection," in Proceedings of the 33rd IEEE Symposium on Security and Privacy, 2012.

[8] Dolan-Gavitt, B., Leek, T., Zhivich, M., Giffin, J., and Lee, W. "Virtuoso: Narrowing the Semantic Gap in Virtual Machine Introspection," In Proceedings of the 32nd IEEE Symposium on Security and Privacy (2011).

[9] H. Inoue, F. Adelstein, M. Donovan, and S. Brueckner, "Automatically bridging the semantic gap using a c interpreter," in Proc. of the 2011 Annual Symposium on Information Assurance, June 2011.

[10] Flavio Lombardi, and Roberto Di Pietro, "KvmSec: a security extension for Linux kernel virtual machines," in Proc. of The ACM symposium on Applied Computing, Honolulu, Hawaii, 2009, pp. 2029-2034.

[11] S. Bahram, X. Jiang, Z. Wang, M. Grace, J. Li, D. Srinivasan, J. Rhee, and D. Xu., "DKSM: Subverting virtual machine introspection for fun and profit," IEEE Symposium on Reliable Distributed Systems, 2010.

[12] M. Zhu, M. Yu, M. Xia, B. Li, P. Yu, S. Gao, Z. Qi, L. Liu, Y. Chen, and H. Guan. "Vasp: virtualization assisted security monitor for cross-platform protection," in ACM Symposium on Applied Computing, pages 554–559, 2011.

[13] Ibrahim, A.S., Hamlyn-Harris, J., Grundy, J., Almorsy, M., "CloudSec: A Security Monitoring Appliance for Virtual Machines in the IaaS Cloud Model," In: Proc. of 2011 International Conference on Network and System Security (NSS 2011), Milan, Italy 2011.

[14] G. Xiang, H. Jin, D. Zou, X. Zhang, S. Wen, F. Zhao, "VMDriver: a driver-based monitoring mechanism for virtualization," in: Proc. of the 29th International Symposium on Reliable Distributed Systems, 2010, pp. 72–81.

[15] PAYNE, B. D. http://google.code.com/p/vmitools, April 25 2012.

[16] B. D. Payne, M. D. P. d. Carbone, and W. Lee, "Secure and Flexible Monitoring of Virtual Machines," in Annual Computer Security Applications Conference, ACSAC, Miami Beach, 2007, pp. 385-397.

[17] Li J, Li B, Wo T, et al. CyberGuarder: A virtualization security assurance architecture for green cloud computing[J]. Future Generation Computer Systems, 2012, 28(2): 379-390