Copyright ⓒ 2018 KSII

A Multi-level Perception Security Model Using Virtualization

Rui Lou[*], Liehui Jiang, Rui Chang and Yisen Wang
State Key Laboratory of Mathematical Engineering and Advanced Computing,
Zhengzhou 450002, China
[e-mail: vivagin@yeah.net]
*Corresponding author: Rui Lou

Received December 4, 2017; revised April 9, 2018; accepted May 24, 2018; published November 30, 2018

Abstract

Virtualization technology has been widely applied in the area of computer security research that provides a new method for system protection. It has been a hotspot in system security research at present. Virtualization technology brings new risk as well as progress to computer operating system (OS). A multi-level perception security model using virtualization is proposed to deal with the problems of over-simplification of risk models, unreliable assumption of secure virtual machine monitor (VMM) and insufficient integration with virtualization technology in security design. Adopting the enhanced isolation mechanism of address space, the security perception units can be protected from risk environment. Based on parallel perceiving by the secure domain possessing with the same privilege level as VMM, a mechanism is established to ensure the security of VMM. In addition, a special pathway is set up to strengthen the ability of information interaction in the light of making reverse use of the method of covert channel. The evaluation results show that the proposed model is able to obtain the valuable risk information of system while ensuring the integrity of security perception units, and it can effectively identify the abnormal state of target system without significantly increasing the extra overhead.

Keywords: Virtualization security, threat model, information perception, VMM protection, anomaly detection

This work was supported by the Natural Science Foundation of China (No. 61572516).

1. Introduction

Virtualization has provided a new and effective way to protect the OS due to its particular advantages such as environmental isolation, underlying control, resource integration, dynamic configuration and so on. It can provide many kinds of security services, for example, the intrusion detection, access control, system kernel protection, etc. It is a hot topic to utilize the virtualization technology in research of system security.

With its advantages in protecting computer OS, virtualization also exposes the system to some new risk. There are mainly four types of security issues in virtualization environment: 1) Internal tempering in virtual machine (VM): the various forms of kernel-level Rootkits which can steal information and break the system in the state that is not easily perceived [1,2]. 2) VM-based rootkit: SubVirt [3], BluePill-like [4] and Vitriol [5]. 3) VM covert channel: including the load-based covert channel [6], the sharing memory covert timing channel [7], the L2 cache covert channels [8] and some other types [9,10,11]. 4) Attacks against VMM: including maliciously modifying and nesting the VMM [12].

To address the security problems existing in virtualization system, some studies combine virtualization with other protection technologies in security design and implementation. Tupakula et al [13] proposed an intrusion detecting architecture taking into account the security policies of virtual domains as well as the specific features of the virtual machines to detect the attacks, with the important components of policy enforcement and behavior capture engine integrated into the VMM. CTVM [14] is a kernel protection framework that adopts the hardware-assisted virtualization technology of extended page table to create the isolated operating environment for untrusted modules, and then monitors them during the runtime to protect the kernel integrity of guest VM. Zhang et al [15] presented a scheme of embedded trusted computing environment based on virtual machine architecture named QEMU, wherein the functions of trusted cryptographic modules are simulated by software, thus with no other additional hardware. Kumara et al [16] presented a virtual machine introspection (VMI) based on malicious process detection approach for VM, which extracted the high level information such as system call, opened known backdoor ports from introspected memory to identify the spurious process. Win et al [17] introduced the mandatory access control (MAC) to protect the in-VM monitoring and the hypervisor, aiming at finally protect the guest VM against attacks with relatively ideal overhead.

Although the methods mentioned above have increased the security of OS by drawing support from virtualization technology, there are still some deficiencies to be improved. Firstly, with insufficient considerations on the particularity of virtualization system in aspects of execution mode, data flow and address isolation, risk models of virtualization system are built too simply to describe the threats it faces [18,19]. Secondly, with the appearance of VMM attacking techniques such as the direct memory access coverings [20] and nested virtualization [21], the assumptions that the VMM and security perception units are safe enough, which most current designs of protective structure generally make, are no longer tenable. Thirdly, the combination between security mechanism and virtualization technology is not fully integrated. With the lack of security channel, the existing interactive way between guest VM and VMM is single and easy to be detected and utilized by malicious program [22,23], which reflects the ineffective cooperation between VM and VMM.

On the basis of former research [24], this paper designs and extends the function of I/O process machine (IOPM) in collaborative-VMM (CVMM) to construct the security management domain (SMD), and then builds a multi-level perception security model in virtualization system aiming at dealing with the problems mentioned above. Related work is firstly listed in Section 2. Then the attack model is constructed based on the analysis of risk assumptions in Section 3. The specific design and implementation of the security model is explained in Section 4. Section 5 carries out the evaluation of the model and Section 6 draws the conclusion of the paper.

2. Related Work

This paper studies the threats of malicious activities to virtualization system itself including VM and VMM, then focuses on the design and implementation of a security protection model based on virtualization, which can be categorized as the research of VM intrusion detection. The main idea of intrusion detection based on VM is to depart the security perception unit from target system using the isolation property of virtualization so that the unit can monitor the intrusion behavior independently, for instance the HyperSpector [25], which reduces the risk points in distributed system by applying virtualization. The current research mainly focuses on the following aspects:

1) Logging and Auditing. For the attackers can modify the syslog in traditional system when they get the control of system, it is no longer reliable, depending on which the intrusion detection system can do the analysis of malicious program. Dealing with the problem, ReVirt [26] gets the logger to operate in a VM isolated from other processes, thus equips the VM with the ability of auditing. The advanced auditing adopts the similar idea to establish a privileged VM to collect audit logs for each protected VM [27]. Oikawa et al [28] proposed the framework that performs simultaneous logging and replay in two isolated VMs running on the same host.
2) Decoy and Interference. With lower costs and risks comparing to physical honeypot, the honeypot system constructed in VM can log the completed activities of malicious program in the isolated position. It is also capable of quickly recovering relying on the backups even if the VM has been broken into. However, the virtual honeypot can be perceived by attackers because it cannot provide the completely real environment of system, and the information provided by a single or multiple virtual honeypots is not enough to reflect the whole feature of intrusion. For the former, the highly interactive honeypots are proposed to give attackers the adequate feedbacks of information [29,30]. For the latter, the distributed honey-net system is set up to provide more accurate attacking report under circumstance of highly cooperating between virtual honeypots [31,32].
3) Out-of-VM Monitoring. Integrating the advantages of host-based and network-based intrusion detection systems, some monitoring systems are proposed to make the intrusion detection components do the monitoring work out of target VM, thus the components can possess a higher security level. However, two problems that have driven the study on modified out-of-VM monitoring are the extra cost produced by exchanging the control when monitoring the calls, and the semantic gap between VMM and target OS which makes the internal events of VM hard to be comprehended by detection component. To solve these two problems, NEM adopts a kind of structure called secure in-VM monitoring to keep the performance advantages [33]. Virtuoso uses a novel approach for automatically creating introspection tools

for security applications [34]; and TxIntro implements the detecting and identifying about target VM merely by retrofitting hardware events [35].

3. Threat Model

3.1 Assumption

Virtualization deepens the software stack in the vertical direction of the system. It changes not only the way OS accesses the resources, but also the trusted computing base (TCB) and executing mode of original system. Purely from the perspective of the abstract function that virtualization provides, it could be found that the use of virtualization does not improve the security of computer system because the VM only represents as the logic example of the underlying physical machine, the threat that the traditional computer system faces still exist in the VM. Besides, it can be seen, by the analysis of the security issues existing in virtualization system in section 1, that the attacker could have more objects to attack. In other words, not only the OS itself but also the VM, VMM and privileged VM are all in risk of being attacked. Since the virtualization system suffers a greater risk of attack, a higher security requirement is needed. Taking comprehensive consideration of the security risks and the minimum conditions under which the proposed mechanism could run, we make assumptions of the threat and establish the attack model of CVMM. According to the basic security structure shown in **Fig. 1**, CVMM, whose TCB contains the hardware and SMD, provides three forms of security services—integrating within VMM, deploying inside SMD and operating in VM system in a front-end way.

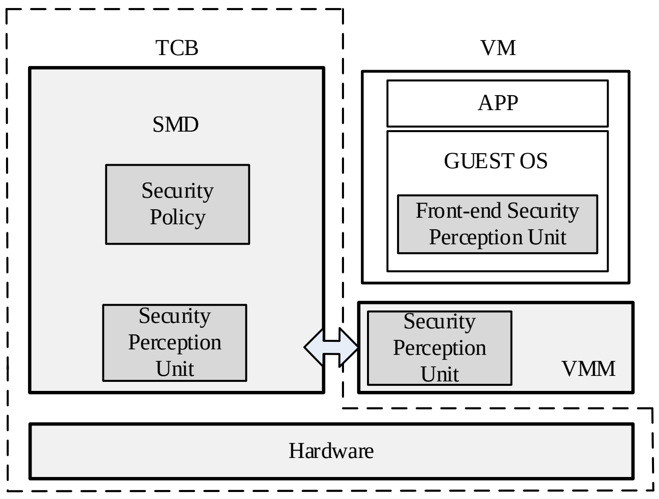


Fig. 1. Basic security structure of CVMM

Assumptions of security preconditions and possible risks for CVMM are made as follows:

1) CVMM's TCB consisting of the hardware and SMD, excluding VMM, is safe enough. The security services in the model would fail if TCB is destroyed.
2) Normally, the data structures in guest OS are reliable depending on which VMM and SMD can conclude the state of guest system. Even if the guest is invaded and the data structures are destroyed, and consequently the malicious programs bypass the security policies relying on them and influence the ability of VMM and SMD that

monitoring information inside the guest, the security of VMM and SMD is still guaranteed and they can continue to scan the key memory and capture the device access operations.

3) By means of utilizing the covert channel, the way of perception between guest VM and VMM has no obvious abnormal features that can be perceived by malicious programs so that it would not trigger the escape and anti-detection behaviors of them.
4) With the threats of illegal modifications by malicious guests and VMBR attack, VMM itself is not safe.
5) SMD is a tidy Linux system and possesses the smaller TCB because it has swept away an amount of irrelevant code. Isolated from VM and VMM, SMD would not be influenced by malicious programs.
6) SMD is in running state all the time. And any operation on key area of VMM's memory would be captured by it to ensure safety of VMM.

3.2 Attack model

As shown in **Fig. 2**, the data flow transmission in CVMM would go through four parts.

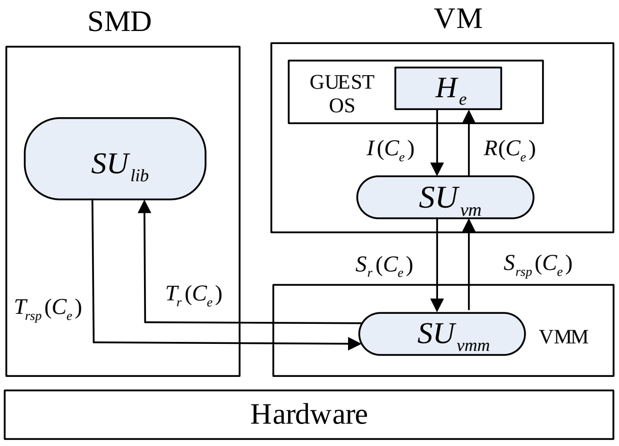


Fig. 2. Data flow in CVMM

The assumptions are made that the security perception unit inside VM—SU_{vm} consists of the code and data, while the library it relying on during operating is defined as SU_{lib}. SU_{vm} is used to initiatively monitor the events E happening in internal VM. The hook H_e has been set up between the start and the end point so that any event e∈E would be intercepted to transmit the control flow to perception unit. Process switching or inter-domain communication can implement this transmission. The symbol C_e expresses the context information of guest OS that H_e captures, while $I(C_e)$ refers to the informing and calling to SU_{vm} initiated by H_e. SU_{vm} sends the processing request $S_r(C_e)$ to VMM perception unit SU_{vmm}, and after that the policy request $T_r(C_e)$ would be delivered to SU_{lib} for decisions. In the reverse direction of data flow, $T_{rsp}(C_e)$ expresses the responding from SU_{lib} and $S_{rsp}(C_e)$ answers the request sent by SU_{vm}. Finally, SU_{vm} responds to hook H_e with $R(C_e)$ which must be enforced including the update of system state and modification on execution flow.

The risks may exist in each procedure during the data and control flow delivery. **Fig. 3** indicates the possible cases that may be threatened and attacked as shown by dotted lines.

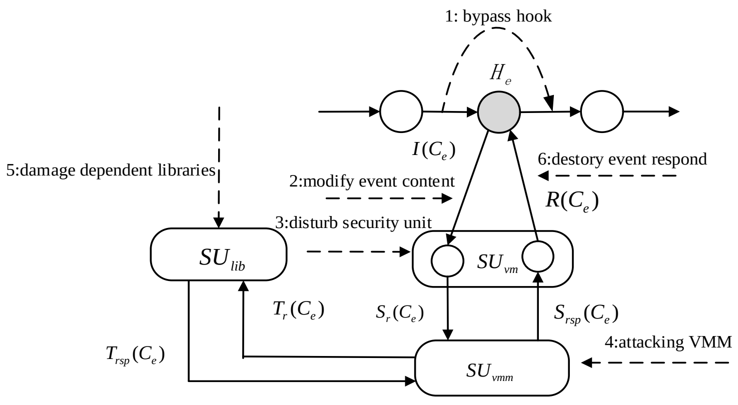


Fig. 3. Attacking model of CVMM

The risky links that are vulnerable to malicious programs are as follows:

1) Bypassing H_e which makes $I(C_e)$ unable to be called.
2) Modifying C_e while it is being delivered to SU_{vm} so that SU_{vm} obtain the error messages.
3) Disturbing SU_{vm} by directly attacking it.
4) Attacking SU_{vmm} by malicious means of DMA covering or VMBR.
5) Damaging SU_{lib} which leads SMD to make wrong decisions.
6) Destroying $R(C_e)$ so that H_e cannot execute the correct feedback.

4. Design and Implementation

This section presents the design and implementation of the key points from section 4.1 to 4.3, and then describes and analyzes the entire security model in section 4.4.

4.1 Enhanced isolation mechanism

Address space isolation, for example the isolation among processes or VMs, or between VM and VMM, is a fundamental measure to protect the system's normal units against malicious behaviors. It primarily pays attention to preventing the address space from illegal border-crossing. From the perspective of information security, an enhanced isolation mechanism is required to realize the more fine-grained address mapping and access authorization configuration.

As section 3.1 describes, both the guest VM and VMM are exposed to potential risks. The components that need to be protected contain the perception units, VMM jump code and SMD. Due to the cross-deployment of risk and security components shown in **Table 1**, it is required to enhance the restriction of address mapping and access authorization among these components.

Table 1. Applications in each class

Insecure cross-deployment	Performance	Isolation measure
Deploying the security unit in risk environment	Setting up perceiving point in guest VM	Prohibiting modifications on security unit
Deploying the risk code in secure environment	VMM code that mapping to SMD space	Banning execution of risk code
Deploying the irrelevant code in risk environment	SMD code that mapping to VMM space	Prohibiting the mapping of irrelevant code

The isolation measures should meet the following constraints:

1) For preventing VMM from damaging SMD's integrity, the address space of SMD cannot be accessed by VMM.
2) SMD can walk through the entire address space. It possesses the reading and writing privileges on VMM code mapped to its space, but no executing permission. Thus SMD is able to read and operate the related contents in VMM (for VMI), and avoid the influence from VMM code at the same time.
3) The perception unit possesses only the reading and executing authorities in VM's address space to prevent from being modified by malicious code in VM, while it maintains all authorities in VMM's address space.
4) VMM jump code cannot be modified by forbidding the writing authority on its own in VMM's address space. SMD is able to conduct the reading and writing operations on VMM jump code mapped to its address space, without any executing authority. It can prevent VMM jump code from disturbing SMD itself.

Setting the page properties of the components with R (reading), W (writing) and X (executing) to satisfy the constraints listed above, we establish the enhanced isolation mechanism to ensure the safety of security units by controlling the address mapping and page property authorizing. **Fig. 4** expresses the mechanism in detail.

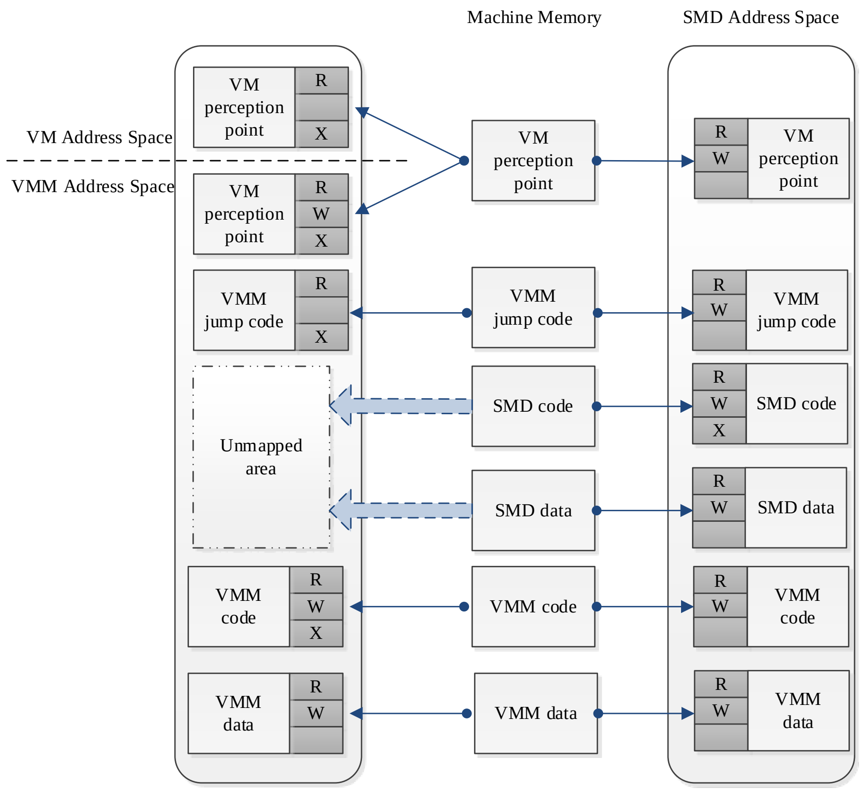


Fig. 4. Schematic diagram of the unit isolation and authority setting

4.2 VMM protection

The guest VM running on VMM or malicious programs activating inside VMM may operate the key areas of VMM kernel such as page tables, interrupt descriptor tables and VM management structures. To protect the integrity of VMM's memory, it is required to make VMM's memory impossible to be modified by attack codes. Since attack codes run in (or start from) non-privileged mode, the modifying operation on VMM can only be accomplished in privileged mode—that is to say, all the operations targeting at VMM's memory must be carried out by VMM's own routines, for instance the initialization and exit handling. Besides, the code and data of VMM cannot be accessed by peripherals in DMA way to defend against the DMA attacking.

It is assumed that virtualization system can be defined as $S = (V, P, M)$ in which V, P and M respectively represent VMM, malicious program and VMM's memory. First of all, S must possess three properties—control flow integrity, modularity and atomicity. The control flow integrity denotes the consistency between source code semantics and actual flow when V is processing. The modularity expresses that V is composed of the initialization function *init*() and a series of exit handling routines $ehr_1(),\ldots,ehr_k()$. The atomicity refers that *init*() will run independently before other programs, as well as the routines $ehr_1(),\ldots,ehr_k()$. The three properties have been kept during the design and implementation process of CVMM. Therefore it possesses the sequential character *Seq*(*S*), which means that there would be four types of states appearing in exclusive mode during running process of *S*—that is to say, *S* executes either initializing and exit handling procedures in privilege mode, or the guest VM and attacking codes in non-privilege mode. The state transition of *S* is shown in **Fig. 5**.

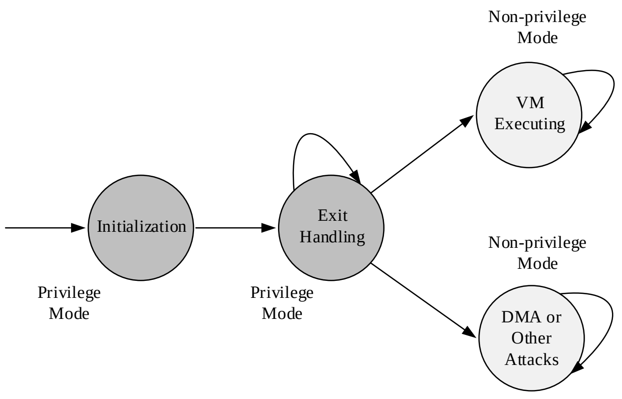


Fig. 5. State transition of *S*

Here are two sequential programs defined as *m* and *n*, and the symbol * expresses the associated and exchangeable relationships, thus *m***n* refers that *m* and *n* are running in nondeterministic order. The context of a certain program can be defined as the symbol & and *m*(&) represents that *m* is executing in context. The definitions of sequential character *Seq*(*S*) and memory integrity protection are given as follows:

Definition1. (Sequential character) *Seq*(*S*) is composed of the sequential programs:

$$init(\&); while(true)\{P(\&) * ehr_1(\&) * \cdots * ehr_k(\&)\}$$

P(&) in the expression above stands for the execution of any attacking code.

Definition2. (Memory integrity protection) The memory integrity of VMM can be ensured, if and only if the condition $\psi(M)$ can be satisfied that all memory under $\psi(M) \equiv M$ is read-only in non-privilege mode.

Theorem1. The memory integrity of *S* can be guaranteed, if and only if both of the following conditions can be satisfied:

$Cond1.\ init(\&)\ satisfies\ \psi(M)$.

$Cond2.\ for\ i \in [1,k],\ ehr_i(\&)\ satisfies\ \psi(M)$.

Proof of sufficiency:

If the memory integrity of VMM has been guaranteed, *M* can be modified only in privilege mode. Assuming that neither *Cond*1 nor *Cond*2 is satisfied, there must exist a certain variable (or code) *v* which leads *init*(*v*) or $ehr_i(v)$ not to satisfy the $\psi(M)$ so that *M* can be modified in non-privilege mode, which is in contradiction with the premise. So the sufficiency is proved.

Before the proof of necessity is given, the assumption that *S* possesses the property of MAC that is built in M has been made firstly. As previously stated, any access to *M* by malicious program *P* in non-privilege mode will trigger the routines of exit handling because of violating the condition $\psi(M)$. Therefore, *P* will satisfy $\psi(M)$ if *S* possesses MAC property.

Proof of necessity:

If *Cond*1 and *Cond*2 are both tenable, *init*(&) and $ehr_1(\&) * \cdots * ehr_k(\&)$ will satisfy the condition $\psi(M)$. Simultaneously assuming that *S* possesses MAC property, *P*(&) also conforms to $\psi(M)$. According to definition 1 and inductive method, $\psi(M)$ ought to be satisfied by the sequential programs $init(\&); while(true)\{P(\&) * ehr_1(\&) * \cdots * ehr_k(\&)\}$. It means that *S* can satisfy the condition $\psi(M)$. So the necessity is proved.

It can be seen from the proof above that the memory integrity of *S* can be ensured if it possesses MAC property. What's more, MAC property requires that the memory access in non-privilege mode should be limited by MAC mechanism supported by hardware and be stored in data field of VMM or other privileged systems. In hardware-assisted virtualization, VM exit is an architecture event that cannot be bypassed, thus provides a very natural MAC condition.

This paper establishes the perceiving mechanism from SMD to VMM based on the distinctive parallel structure of CVMM that makes SMD can perceive the event modifying the VMM's memory and verify its validity independently even if VMM itself has been attacked. The main idea of the mechanism is to set the VMM's kernel area to read-only property by using page protection mechanism, which will lead to the page fault when modifying on that area occurs. By expanding the mechanism of informing SMD into initial page fault handlers, SMD can verify the modification event to decide whether it is normal during the pause of VMM before SMD returns the decision. **Fig. 6** shows the perceiving mechanism to protect VMM.

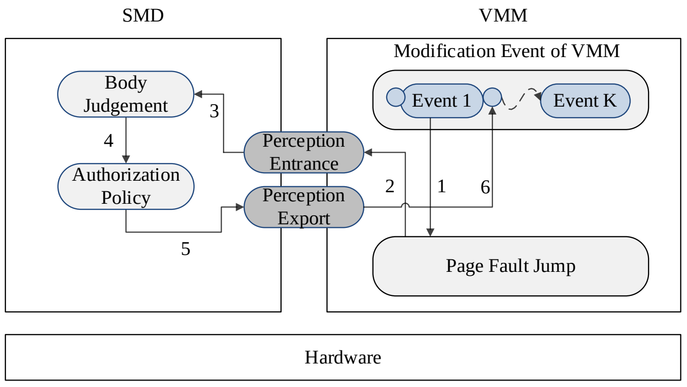


Fig. 6. Protection of VMM's memory integrity based on parallel perceiving

The key to implement VMM protection is the generation of the page fault events and the detection of their types. Considering that the address translation must be performed by shadow page table (SPT) when VM needs to access memory, we set R/W bit of all page table entries to 0 when SPT is created, so as to prohibit all the writing operations on memory (including legal memory area belonging to VM and VMM memory area). Thus, a large number of page fault events will occur from VM's initial running stage on which lead VM to exit and transfer the control to VMM, while SMD also has the chance to do further analysis of the events. Due to the writing protection policy and the unsynchronized features of SPT entry and VM's page table entry, there exist four types of page faults during the running process of VM, which need to be handled differently:

1) The page fault caused by the first legal writing operation on VM's legal memory area because of the violation of writing protection. In this case, the corresponding R/W bit will be set to 1 so that guest VM could continue to do the writing operations.
2) The page fault caused by the legal operation whose corresponding target SPT entry is empty. Due to swapping memory pages of VMM or TLB flush operation, the SPT entries corresponding to legal memory of guest VM may be deleted, which will lead to page fault when writing operations on these areas occur. For this page fault type, the corresponding entry will be added in SPT.
3) The page fault caused by missing pages within guest VM. Similar to type 2), a page fault will occur because of swapping memory pages of VM or TLB flush operation. For this type, the guest VM performs the conventional page fault handling.
4) The page fault caused by malicious writing operations which violate the writing protection. Owing to the writing protection strategy, the first writing operation of malicious program will still be in violation of this strategy, even if it has added the page table entry to SPT that could make itself able to access VMM memory in some way. Focusing on this type, we'll replace the instruction corresponding to Guest.EIP with NULL, and then returns to VM's execution; otherwise the VM will repeatedly execute the instruction and continue to generate page fault events.

As the detection of page fault type is achieved and supported mainly by accessing to both SPT and VM page tables, SMD should have the ability to acquire them. Considering that SPT and VM page table are respectively pointed to by CR3 in VMM address space and in VM address space, we parsed the virtual machine control structure (VMCS) to enter into the host-state area and guest-state area under VMCS, where the Host_CR3 and Guest_CR3 can be acquired.

4.3 Secure perception between VM and VMM

It is beneficial for system security to improve the CVMM's ability of self-protection by combining the advances of VM's identification and VMM's supervision. Before attacks are actually generated, the malicious program will present some unusual behaviors such as altering or deleting files, creating processes and so on. This stage can be called threat-upstream. In this stage, security unit deployed in guest VM is able to perceive the abnormal activities in advance. Afterwards, the malicious program has to operate the underlying resources mapped form the seniors when they launch attacking, for example the access to virtual memory and device. This stage correspondingly can be called threat-downstream. Since the operations have triggered the preset conditions of VM exit, VMM can capture them to do the further analysis in this stage.

The method of protection based on perception is applied as follows: the guest VM reports the suspicious programs to VMM in threat-upstream; and then VMM obtains the information to do the follow-up processing in threat-downstream. An important condition shall be satisfied at first that VM can send the information to VMM in a way as covert as possible. According to the classical descriptions about existence conditions of covert channel [36], this paper summarizes the basic requirements to build storage channel combining with the properties of CVMM. And all of them have been met during the implementation.

First, both VMM and the guest VM shall have the ability to access some particular chunks of shared memory. To meet the requirement, a certain memory area R is partitioned by modifying the VM's and VMM's page tables and memory management algorithms that locates in physical address space. After being set as reserved area in memory management algorithms, R can be accessed by both VM and VMM but cannot be allocated by them, as shown in **Fig. 7**.

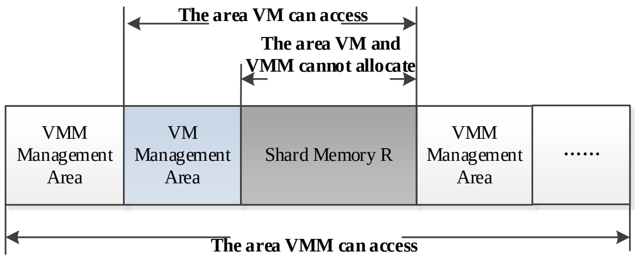


Fig. 7. Diagram of shared memory R

Secondly, the guest VM can modify the data in R. As the shared memory R is established, the guest VM has already possessed the ability to access and operate R.

Thirdly, VMM must be capable of perceiving the change of data in R that is the key point of building storage channel. With the help of the instruction VMCALL provided by the processor that support virtualization, VM can actively hand over the control to VMM so that VMM can make deeper study on the events happening in VM. During the normal running of guest VM, VMCALL will not be triggered unless VM needs to communicate with VMM. Once detecting VMCALL, VMM will take out the secret information from R written by VM. This process can be shown as **Fig. 8**.

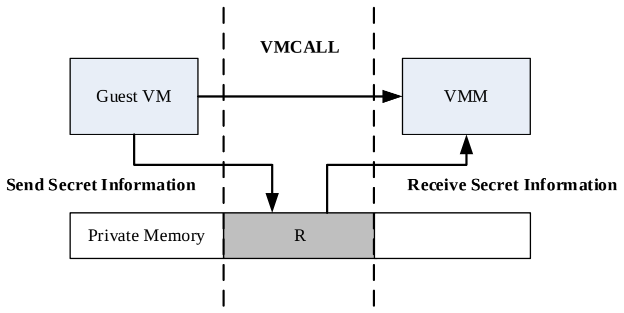


Fig. 8. Storage covert channel between VM and VMM

Finally, the synchronization mechanism is necessary to ensure the correctness of sending and receiving information. Some certain events can lead to the exit of VM which will pause until it gets feedbacks—that is to say, VM will not continue to send information until VMM have completely received it. Therefore, this requirement can be easily satisfied in virtualization system.

As mentioned above, this paper has established the secure perception mechanism between VM and VMM by making reverse use of the method of covert channel. The operating process of the mechanism can be described as follows. When the guest VM perceives the abnormal operation, the security perception unit will not directly clean it up to avoid triggering the anti-detection behavior of the malicious program. Now, the guest VM writes the particular information such as process identifier (PID) or extended stack pointer (ESP) to shared memory R and then executes VMCALL instruction to make the control trap into VMM. After that, VMM takes out the information and delivers it to SMD to do further analysis. Finally, SMD returns the processing result to VMM that will directly ruins the low-level resources corresponding to the malicious program afterwards. The whole mechanism can be shown as **Fig. 9**.

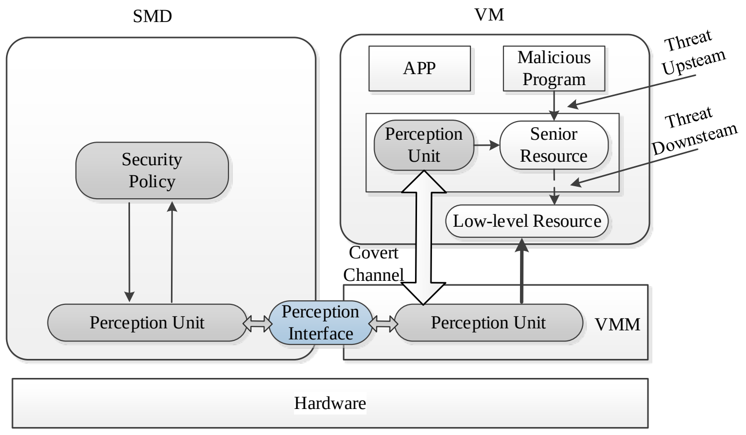


Fig. 9. Diagram of covert perceiving mechanism

The target of secure perception is to handle the malicious program in VM. The precondition is to comprehend the high-level semantics of guest VM, which SMD should restore from the information it acquires. We chose the Linux release version Fedora 13 as the VM system of which the process information is maintained by the structure task_struct. So the secret information transmitted from secure channel provided by VM ought to make task_struct be parsed. As the system stack of process and the structure thread_info locate in the continuous two pages and ESP points to the top of system stack, we can calculate the location of thread_info through ESP and then leverage the thread_info()->task to find task_struct. Considering that SPT of the process maintains the mapping relation from guest virtual address to host physical address, we can locate the ESP's position in machine memory according to the transmitted ESP and mapping relation, and then parse the thread_info and task_struct to acquire three key structures: process context, signal queue and the virtual address space, as **Fig. 10** shows.

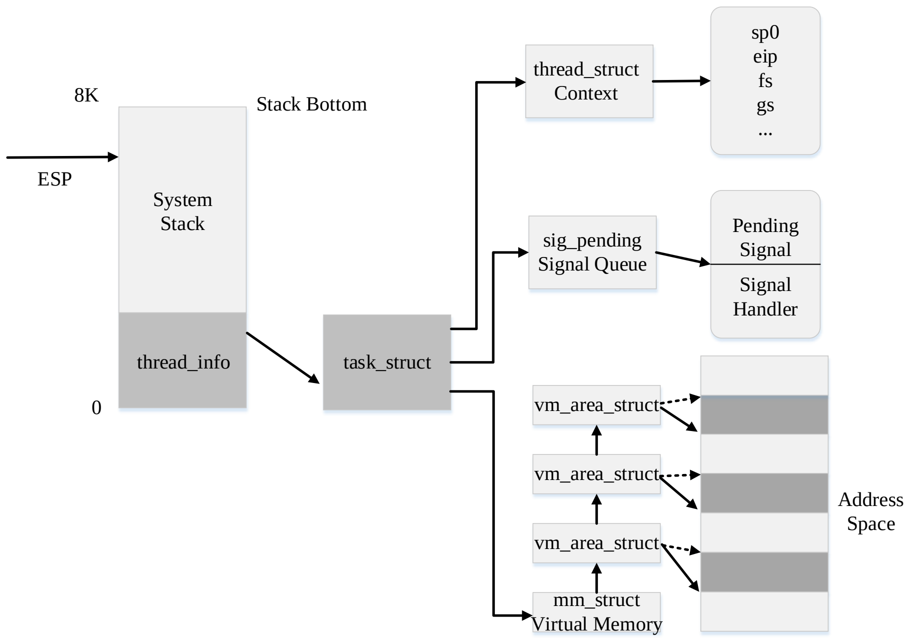


Fig. 10. Restoration of high-level semantics of target process

On the basis of obtaining the key information of process, we implement three types of processing strategies: ①Modifying thread.eip of task_struct to make it point to the invalid address or to a piece of code containing the terminating operation, which leads guest OS itself to terminate the process; ②Adding a termination signal SIGKILL to signal queue so that OS could naturally terminate the process; ③Covering the address space of process forcibly from VMM layer in case of invalidation of previous two strategies. The vm_area describes the linear address space of process, so we could leverage task_struct->mm->mmap to locate the position of it in machine memory and then cover the original data, thus eliminate the malicious code fundamentally.

4.4 The multi-level perception security model

The preceding sections have demonstrated analysis and solutions of some security problems in virtualization system: 1) For isolation of security perception units, an enhanced isolation mechanism is set to make risk components possess the minimum legal authorities to protect the units from risk environment. 2) For the invalid assumption of VMM's security, a perceiving mechanism from SMD to VMM based on the distinctive parallel structure of CVMM is established to ensure the integrity of VMM's memory. 3) For the weak capability of interaction between VM and VMM in security systems, a covert storage channel is constructed to make the guest VM capable to send the sensitive information to VMM for further analysis in private. The security protection model this paper designs can be shown as **Fig. 11**.

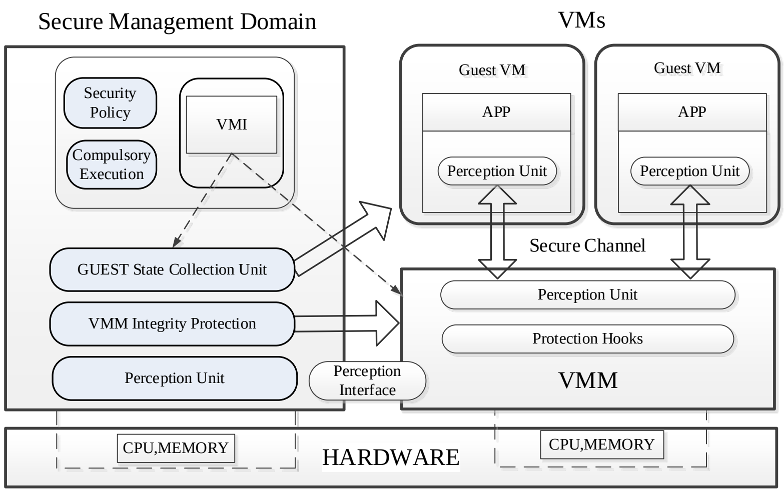


Fig. 11. Diagram of the multi-level perception security model

The structure and characteristics of the protection model are described as follows.

1) In the whole protection system, VMM provides the virtualization capability and SMD offers the security services. VMM and SMD have the same highest privilege. They possess the separate processors and memory, and they are physically isolated by means of resource partitioning.
2) The perception units are respectively deployed in VM, VMM and SMD so that VM is able to initiatively deliver the sensitive information to VMM. Depending on the information, VMM and SMD can block malicious behaviors in a higher privilege.
3) The hooks are set, focusing on the key resources and paths. Therefore, VMM can monitor the activities of malicious program in the view of underlying resources and gain the control in time when the attack on senior resources occur.
4) The units of VMI, security policy and compulsory execution are used to analyze and identify the behaviors in guest VM. If malicious behaviors are figured out, related operations will be prevented from executing.
5) With the same privilege as VMM, SMD can access all the memory and enable VMI unit to get the state of VM directly to reduce the performance overhead caused by VMM's frequent interventions in the guest VM.
6) The unit of VMM integrity protection in SMD can ensure VMM's safety by verifying the events of memory modifications.

The proposed model can effectively enhance the capability of risk resistance of the virtualization system. The enhanced isolation mechanism can ensure that perception unit in VM space would not be modified, thus prevent it against the risk of being tempered with by malicious program. It also can block out the interference that SMD suffered from VMM, and make it provide sustained and effective security services. The VMM protection mechanism ensures the integrity of VMM, including the security of perception unit in VMM, which can effectively guard against the attack risk to VMM. The secure perception between VM and VMM can make VM covertly deliver the key information of malicious program to security components outside and then eliminate it from underlying layer, which can avoid the risk of triggering the anti-detection of malicious program.

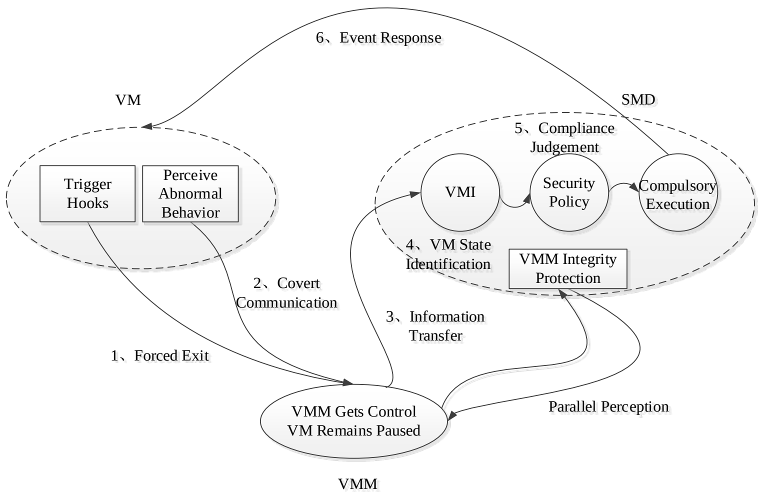


Fig. 12. Execution flow diagram of the security protection model

The execution flow of the protection model is shown as **Fig. 12** and described in the following. Firstly, VMM sets the hooks in the key paths and resources points and they will not be triggered during the normal executing process of VM. However, VMM will gain the control in two cases. One is that the operation in VM has broken the rules of VMM protection that may be generated by malicious behavior. The other is that VM reports the internal information to VMM actively via secure channel when suspicious behavior appears. After gaining the control, VMM sends the acquired information to VMI unit set in SMD via perception interface. The VMI unit can restore the senior semantic information for which the security policy unit will do further verification to identify whether the corresponding behavior conforms to the rules or not. The processing result for the system behavior needs to be enforced by VMM that either allows VM to continue to execute or prohibits the current operation. With the help of the security policy unit, VMM integrity protection unit will monitor the memory operations all through the system's running to identify and then prevent the malicious behavior from tampering with VMM.

5. Evaluation

5.1 Work environment

1. Implementation environment

We have implemented the perception security model shown in Figure 11 on CVMM. The hardware and software environment of implementation is shown in **Table 2** and **Table 3**.

Table 2. Hardware environment

Sort	Description
Processor	Intel Core i5 650M @3.2GHz
Chipset	Intel Q57
Memory	4GB DDR3 1333MHz
Hard disk	500GB 7200 RPM SATA
Graphics	ATI Radeon HD 4650
Architecture	Inter X86

Table 3. Software environment

Sort	Version
VMM	CVMM(Xen-based Sandbox)
IOPM	Fedora 13(kernel: Linux 2.6.38)
VM	ttylinux(kernel: Linux 2.6.38)
Device mode	Qemu-dm 2.2
Security software	ClamAV 0.99.2

This section will carry out functional test on VMM protection and VM protection, and evaluate the system overhead. The test of VMM protection shows that the security system can detect it and make decisions timely when the attack on VMM area occurs to ensure the integrity of VMM. The test of VM protection shows that security mechanism is able to acquire the key information through secure channel and successfully parse out the high-level semantics of malicious process to eliminate it and protect VM. Finally, we evaluate the overhead of these two mechanisms, and then evaluate the performance of perception security model when the proposed two mechanisms are both applied in CVMM simultaneously.

2. Prototype platform description

The proposed model is implemented based on CVMM platform which provides the support of hardware-assisted virtualization technology (VT-x, VT-d, etc.). CVMM is a specific implementation of Xen-based sandbox and a virtualization platform with a small amount of code. The architecture of CVMM is very different from traditional virtualization system, as shown in **Fig. 13**.

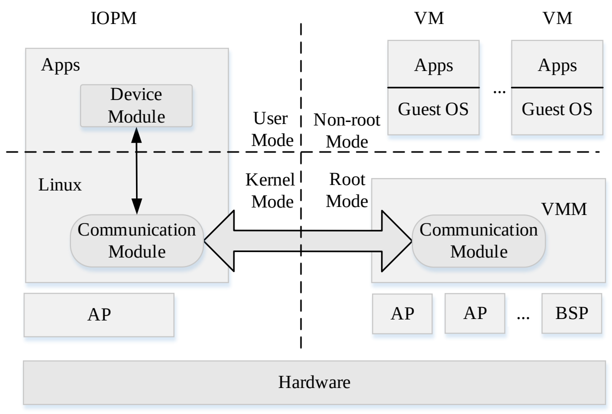


Fig. 13. CVMM architecture

CVMM converts an ordinary VM into a privileged IOPM by the way of hardware partitioning, allowing it to directly control parts of hardware resource. IOPM controls one application processor (AP) and runs a modified Fedora inside. It applies Qemu-dm to complete the device virtualization and provide the virtual resource associated with devices. VMM controls the bootstrap processor (BSP) and other APs, provides VCPU resource for VM and manages the memory allocation of system, and realizes the virtualization of processor and memory. IOPM interacts with the VMM through inter-processor interrupt (IPI) and shared memory. IOPM possesses the same privilege as VMM and directly runs on hardware, forming the parallel collaborating structure. Therefore, it not only avoids frequent privilege switching, but also changes the way of I/O processing that uses Qemu-dm to complete the I/O request of guest VM.

CVMM makes full use of characteristics of multi-processor architecture and hardware virtualization to manage and allocate virtual resource. It can improve the efficiency of the system with reduction of the I/O processing cost.

5.2 Security analysis

1. Resource monitoring

As mentioned in Section 4.3, the malicious program would present some unusual behaviors before attacks are actually generated, and it has to operate the underlying resources mapped form the seniors when they launch attacking. According to the probable relationships between senior behaviors and underlying operations of malicious program [37], VMM needs to focus on controlling the operations for kernel proper memory and monitoring the specific underlying activities of malicious program to protect the key resources of system. It mainly includes four aspects as follows.

1) Writing to sensitive memory. VMM needs to set this kind of memory to read-only mode. The malicious program would be captured by VMM once it is attempt to modify the memory.
2) Executing system call. Quick system calls and software traps are the two ways to generate system calls. For doing further analysis of malicious program, VMM shall set the service entry address to invalid one to lead VM to exit when system call is triggered by the program.
3) Context switching. Accompanied by page table switching, the context switching can be monitored by VMM through the change of CR3. Afterwards, VMM continues to match the current process with the target and acquires more detailed information by means of VMI.
4) Modifying boot sector. The boot sector is tending to be modified and utilized by malicious program. Through the device virtualization module, VMM is able to set the boot sector non-writable to capture the write operation to this area.

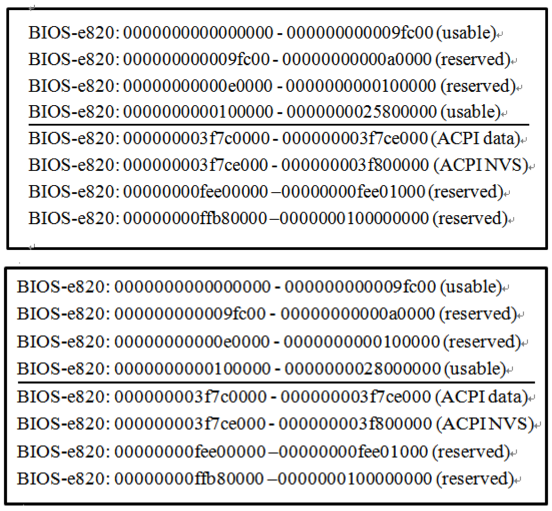
2. Forced transfer of control flow

To protect the resources of guest VM, it is important on the one hand to monitor the key points as listed above so that VMM can gain control once related resources are accessed; on the other, it is also necessary to force the information flow completely transferred in the default path for purpose of more reliable post-processing. For some perceiving sites and transfer points of control flow are deployed in guest VM, not only shall they be isolated from the code of guest VM to prevent malicious program from bypassing or destroy them, but also they need to possess the self-contained property to execute themselves independently of VM code. As analyzed in Section 3, the security requirements of control flow transfer can be expressed in the following.

1) $I(C_e)$ will be activated if and only if e is generated legally and normally.
2) C_e cannot be modified during from e's occurrence to the process of $I(C_e)$.
3) $S_r(C_e)$ and $T_r(C_e)$ cannot be tempered with during the process of delivering C_e.
4) $T_{rsp}(C_e)$ and $S_{rsp}(C_e)$ cannot be tempered with during the process of returning C_e.
5) $R(C_e)$ must be enforced.

5.3 The results of VMM protection

In this section, the mechanism of VMM protection is tested by structuring the VMM loophole to provide programs the chance to access VMM memory. Considering that the VM acquires the whole physical memory depending on E820 of virtual BIOS structured by VMM, it can make VM access the VMM's private memory by modifying the E820 in VM kernel. As **Fig. 14** shows, the guest VM can cross the boundary to access the extra memory size of 40M that belongs to VMM and cannot be accessed by VM in normal state.



```
BIOS-e820: 0000000000000000 - 000000000009fc00 (usable)
BIOS-e820: 000000000009fc00 - 00000000000a0000 (reserved)
BIOS-e820: 00000000000e0000 - 0000000000100000 (reserved)
BIOS-e820: 0000000000100000 - 0000000025800000 (usable)
BIOS-e820: 000000003f7c0000 - 000000003f7ce000 (ACPI data)
BIOS-e820: 000000003f7ce000 - 000000003f800000 (ACPI NVS)
BIOS-e820: 00000000fee00000 –00000000fee01000 (reserved)
BIOS-e820: 00000000ffb80000 –0000000100000000 (reserved)
```

```
BIOS-e820: 0000000000000000 - 000000000009fc00 (usable)
BIOS-e820: 000000000009fc00 - 00000000000a0000 (reserved)
BIOS-e820: 00000000000e0000 - 0000000000100000 (reserved)
BIOS-e820: 0000000000100000 - 0000000028000000 (usable)
BIOS-e820: 000000003f7c0000 - 000000003f7ce000 (ACPI data)
BIOS-e820: 000000003f7ce000 - 000000003f800000 (ACPI NVS)
BIOS-e820: 00000000fee00000 –00000000fee01000 (reserved)
BIOS-e820: 00000000ffb80000 –0000000100000000 (reserved)
```

Fig. 14. Results of VM memory detection before and after modifying E820

At this time, the layout of machine memory can be expressed as **Fig. 15**. The legal access area for VM ranges from 10M to 610M, while the illegal ranges from 610M to 650M.

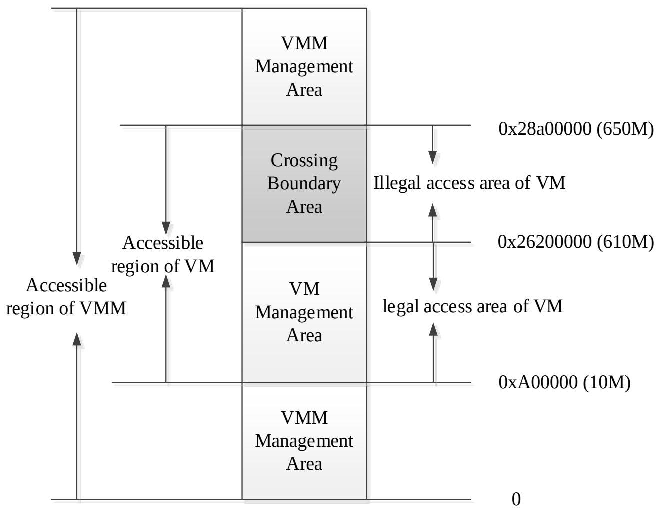


Fig. 15. Layout of machine address after modifying E820

When processes generate the writing requests, the page fault will be triggered so that SMD can judge whether the writing operations are locating on that area size of 40M by analyzing the destination machine addresses in shadow page table. **Fig. 16** expresses the statistic of page faults over a period.

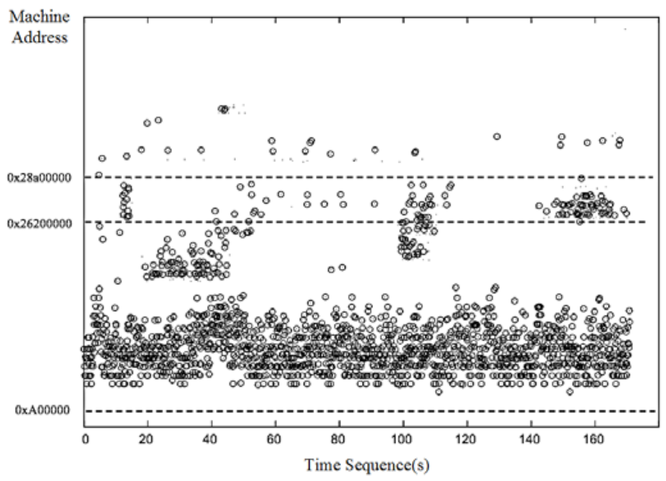


Fig. 16. Page fault distribution in machine address space

The page faults belonging from 0xA00000 to 0x26200000 are legal, while the ones ranging from 0x2620000 to 0x28a00000 are generated by illegal operations of modifying VMM's memory. Besides, the missing pages of VMM itself trigger the page faults locating above 0x28a00000. After recognizing the different types of page faults, SMD will take measures to deal with them according to the default policy. For the page fault caused by operating on VMM's memory, SMD replaces the instruction corresponding to Guest.EIP with NULL, and then returns to VM's execution.

5.4 The results of VM protection

In our proposed system, the prototype of guest VM is the Linux release version Fedora 13 with the kernel version 2.6.38. In order to verify the actual effect of perception mechanism in system protection, malicious programs in Linux system are collected as test data, as shown in **Table 4**.

Table 4. Malicious software used in the perception mechanism test

Name	Behavior description
Slapper	Worm based on the bugs of OpenSSL library
Bliss	Virus that infects and locates ELF files, overlays binary files with malicious code
Staog	Virus that infects ELF files
Typot	Trojan that scans distributed ports and generates TCP packets with window 55808
Mydoom	Worm that launches DoS attack by means of network spread and process termination
TNF	DDoS agent that can launch attacks such as ICMP Flood, Smurf and so on
Lindose	Cross-platform virus infecting both Windows PE and Linux ELF files
ADORE.A	Worm that can rewrite /bin/ps and open port 65535
CHEESE.A	Worm that removes all /bin/sh in /etc/inetd.conf file and close inetd

Taking Lindose as an example, the test method of the perception mechanism mainly includes three steps: ESP transmission, data structure analysis and process destruction:

1) ESP transmission. After starting Lindose, the extended security software ClamAV deployed in VM will first report the threat in system and transmit Lindose's PID to perception unit in VM. Then the unit identifies and records the process corresponding to the PID. When the process Lindose is scheduled by kernel, the perception unit gets ESP from the kernel stack of current

process and writes it to the defined memory R, and then calls VMCALL instruction to generate the VM-exit event. At this moment, the perception unit in VMM can obtain the ESP of Lindose from R, as analysis in section 4.3.

2) Data structure parsing. The obtained ESP value is the process linear address and it must be converted to the corresponding machine address by SPT, which can be used to gradually parse out the corresponding position of thread_info, task_struct and also vm_area in machine memory. The mapping relation between linear address and machine address of key data structure of Lindose is shown in **Table 5**, wherein the symbol ⊗ represents no established mapping from the corresponding linear address to machine address of vm_area in SPT. There are two reasons of this situation: one is that the guest VM does not allocate the actual physical page for corresponding vm_area region, the other is that the SPT and the page of Lindose in guest VM have not been synchronized yet.

3)

Table 5. Mapping relation from linear address to machine address of lindose

Key Structure of Process	Linear Address	Machine Address
ESP	0xdaacff68	0x64179f6
thread_info	0xdaace000	0x6416000
task_struct	0xdab18c50	0x65a9430
vm_area	0x54c000 - 0x56a000	0x72a0000-0x72ce000
	0x56a000 - 0x56b000	0x72ce000-0x72cf000
	0x56b000 - 0x56c000	0x72cf000-0x72d0000
	0x572000 - 0x6f8000	⊗
	0x6f8000 - 0x6fa000	0x748b000-0x748d000
	0x6fa000 - 0x6fb000	0x748d000-0x748e000
	0x6fb000 - 0x6fe000	0x748e000-0x7491000
	0x8048000- 0x804a000	⊗
	0x804a000 - 0x804b000	⊗
	0x9980000 - 0x99a1000	0x7491000-0x74b2000
	0xb78bc000 - 0xb78bd000	⊗
	0xb78dc000 - 0xb78dd000	⊗
	0xb78dd000 - 0xb78de000	⊗
	0xbfca8000 - 0xbfcca000	⊗

4) Process destruction. In the test, we adopt the way of covering vm_area to destroy the malicious program. For vm_area area that has been mapped, the random data or simple 0 is written to the corresponding area of machine memory. And for the area that has not been mapped, we don't bother to deal with it. As the code in Lindose address space has been cleared, Lindose will be terminated when it is scheduled next time because of the generation of invalid opcode exception.

5.5 Overhead

1. VMM protection overhead

The overhead of VMM protection comes from the VM exit caused by page faults and the communication between VMM and SMD. We choose three kinds of benchmark tools – the SpecInt2006, IOZone and SysBench – to evaluate the overhead of VMM protection mechanism. SpecInt2006 is a computer benchmark specification for CPU processing power, and IOzone is a filesystem benchmark tool that generates and measures kinds of file operations, while SysBench is a modular and multithreaded benchmark tool mainly used to evaluate the database load under various system parameters. The three tools are complementary to each other in aspects of test object and method to some degree, and have wide range and precise indicators. We could take advantage of the tools to comprehensively evaluate the overhead of the mechanism from different aspects such as CPU processing capacity, read and write performance of file system and so on. We use them to test and compare the time consumption when enabling and disabling the mechanism of VMM protection. The result can be shown as **Fig. 17** that the time overhead when enabling the protection reaches to almost 9% increases relative to the disable situation.

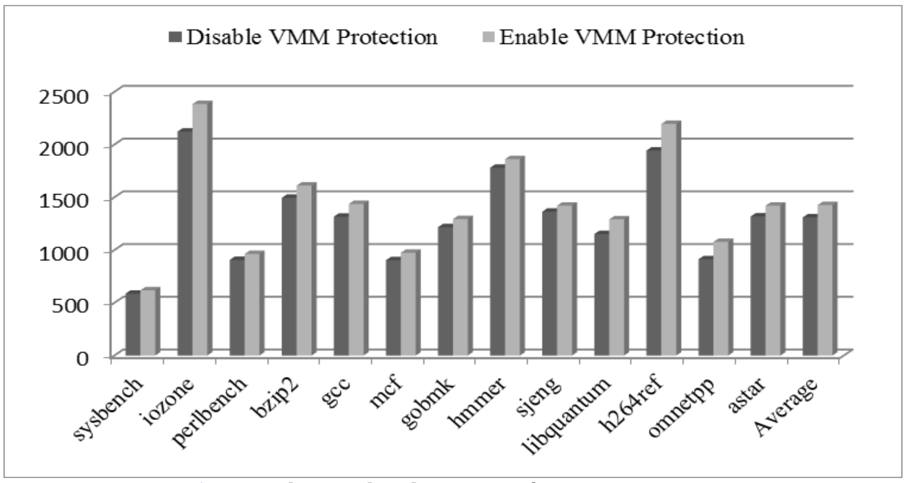


Fig. 17. The overhead statistic of VMM protection

Due to the read-only setting, the writing operation on certain memory for the first time will trigger the page fault that will be enabled during the process of subsequent execution. Therefore, the overhead is mainly generated at the beginning of the program's running. It spends almost 9% extra overhead to complete the VMM protection that has little impact on system performance.

2. VM protection overhead

The cost of perception mechanism is mainly derived from the two aspects of VM exit and process address parsing. In order to evaluate the effect of the two factors on the system performance, the relative cost of the mechanism is tested. Assuming that T_v represents the time cost from starting the malicious process to covering the address space of it, while T_r represents the cost in non-virtualization environment, we focus on the relative cost r that can be expressed as $(T_v - T_r)/ T_r$. **Fig. 18** shows the relative cost of leveraging perception mechanism to process the malicious programs, from which we can see that the average relative cost is about 6.5% with the maximum of 10.5%. Compared with the result in non-virtualized system, the time cost of perception mechanism has not increased significantly.

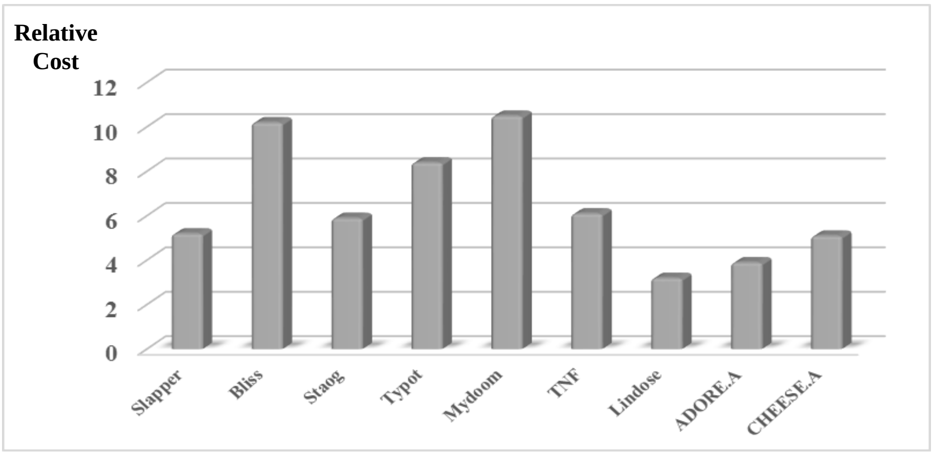


Fig. 18. Bar graph of relative cost of perception mechanism

3. Synthetic overhead

The overheads, generated by six applications run respectively in original system and CVMM, are compared by evaluating the system performance after bringing in the security mechanism. The six applications belong to three different types: CPU-intensive, I/O-intensive and the mixed type. The contrast of resulting overheads is shown as **Fig. 19**.

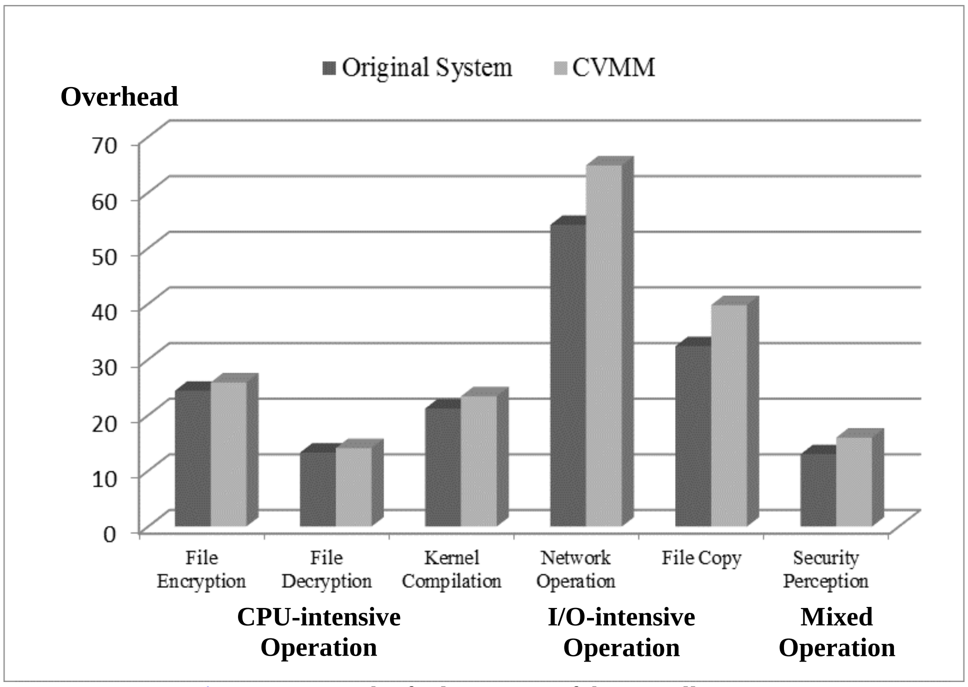


Fig. 19. Bar graph of relative cost of the overall system

Here we also use relative cost *r* to measure the difference of overheads generated between CVMM and original system (non-virtualization system). It can be seen from the figure above that *r* ranges from 6% to 10.4% for CPU-intensive operations which means that the impact of virtualization on normal CPU operations is not significant; while *r* ranges from 19.7% to 22.8% for I/O-intensive operations because the I/O operations in virtualization system need to be intervened by VMM that deliver them to device model. And for mixed operations, *r* is about 23.1% due to the frequent switching between the address space of VM and VMM in virtualization environment.

6. Conclusion

In this paper, we have proposed a multi-level perception security model using virtualization. We start with creating the threat model based on the analysis of security risk that CVMM faces. By setting the asymmetric address mappings and access permissions, we can either isolate the security components and perception points from easily infected code, or prohibit them from being modified. To protect the integrity of VMM, we establish the perceiving mechanism from SMD to VMM based on the construction of page fault events and identification of their type. The change of VMM memory can be perceived and verified by another system possessing the same privilege, the SMD. For further protection of guest VM, we implement the secure perception combining the properties of guest VM and VMM. The guest VM is able to transmit the key information in a secure way with the support of hardware-assisted technology. SMD can successfully restore the advanced semantics of the target process from the low-level information it acquires which overcomes the difficulty of semantic gap. The evaluation results show that the proposed security model could obtain the valuable information of target system while the integrity of security perception units is ensured, and it also could prevent VMM from being modified maliciously and identify the abnormal state of target system effectively while the extra overhead does not increase significantly.

However, there are some restrictions of the proposed model to be broken through. The identification of unknown abnormal behavior remains to be further improved. As we adopt the write protection strategy of SPT to protect VMM memory, it will cause a large number of page faults when the process first performs write operation. The VM exit handling and communication between VMM and SMD will generate the considerable overhead, which may also be detected and used by malicious program. In future work, we plan to introduce the appropriate intelligent algorithms into our model to classify the intrusion behavior, and thus to improve the identification ability and accuracy of the security mechanism. Moreover, we consider to set the dynamic strategy of write protection according to real-time behavior of target VM, so as to decrease the number of VM exit operation caused by page fault, and finally leads to reduce the performance overhead of VMM protection.

References

[1] L. Zhang, X. Chen, Y. Ren and H. Li, "Kernel-level rootkit detection technology based on VMM," *Netinfo Security*, vol 4, pp. 56-61, 2015. Article (CrossRef Link)

[2] L. Zhang, S. Shetty, P. Liu and J. Jing, "RootkitDet: Practical end-to-end defense against kernel rootkits in a cloud environment," in *Proc. of European Symposium on Research in Computer Security (ESORICS)*, pp. 475-493, September 7-11, 2014. Article (CrossRef Link)

[3] S. King and P. Chen, "SubVirt: Implementing malware with virtual machines," in *Proc. of IEEE Symposium on Security and Privacy (S&P)*, pp. 314-327, May 21-24, 2006. Article (CrossRef Link)

[4] D. Anthony, E. Filiol and I. Lefou, "Detecting (and creating!) a HVM rootkit (aka BluePill-like)," *Journal in computer virology*, vol 7, no. 1, pp. 23-49, 2011. Article (CrossRef Link)

[5] B. Robert, J. Vetter and J. Nordholz, “The threat of virtualization: Hypervisor-based rootkits on the ARM architecture,” in *Proc. of International Conf. on Information and Communications Security (ICICS)*, pp. 376-391, April 5-7, 2016. Article (CrossRef Link)

[6] O. Keisuke and Y. Oyama, “Load-based covert channels between Xen virtual machines,” in *Proc. of ACM Symposium on Applied Computing (SAC)*, pp. 173-180, March 22-26, 2010. Article (CrossRef Link)

[7] J. Wu, L. Ding, Y. Wang and W. Han, “Identification and evaluation of sharing memory covert timing channel in Xen virtual machines,” in *Proc. of IEEE International Conf. on Cloud Computing (CLOUD)*, pp. 283-291, July 4-9, 2011. Article (CrossRef Link)

[8] Y. Xu, M. Bailey, F. Jahanian, K. Joshi, M. Hiltunen and R. Schlichting, “An exploration of L2 cache covert channels in virtualized environments,” in *Proc. of ACM workshop on Cloud computing security (CCSW)*, pp. 29-40, October 21, 2011. Article (CrossRef Link)

[9] Z. Wu, Z. Xu and H. Wang, “Whispers in the hyper-space: high-bandwidth and reliable covert channel attacks inside the cloud,” *IEEE/ACM Transactions on Networking (TON)*, vol 23, no. 2, pp. 603-614, 2015. Article (CrossRef Link)

[10] Y. Lin, S. Malik, K. Bilal, Q. Yang, Y. Wang and S. Khan, “Designing and modeling of covert channels in operating systems,” *IEEE Transactions on Computers*, vol 65, no. 6, pp. 1706-1719, 2016. Article (CrossRef Link)

[11] P. Ranjith, C. Priya and K. Shalini, “On covert channels between virtual machines,” *Journal in Computer Virology*, vol 8, no. 3, pp. 85-97, 2012. Article (CrossRef Link)

[12] H. Nemati, S. Sharma and M. Ragenais, “Fine-grained nested virtual machine performance analysis through first level hypervisor tracing,” in *Proc. of IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pp. 84-89, May 14-17, 2017. Article (CrossRef Link)

[13] U. Tupakula, V. Varadharajan and D. Dutta, “Intrusion detection techniques for virtual domains,” in *Proc. of International Conf. on High Performance Computing (HiPC)*, pp. 1-9, December 18-22, 2012. Article (CrossRef Link)

[14] L. Zhang, X. Chen, L. Liu and H. Li, “A kernel integrity protection technology based on virtual machine,” *Journal of University of Electronic Science & Technology of China*, vol 44, no. 1, pp. 117-122, 2015. Article (CrossRef Link)

[15] L. Zhang and X. Kong, “Embedded trusted computing environment build based on QEMU virtual machine architecture,” in *Proc. of International Symposium on Computational Intelligence and Design (ISCID)*, vol 1, pp. 193-196, December 13-14, 2015. Article (CrossRef Link)

[16] M. Kumara and C. Jaidhar, “Virtual machine introspection based spurious process detection in virtualized cloud computing environment,” in *Proc. of International Conf. on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE)*, pp. 309-315, February 25-27, 2015. Article (CrossRef Link)

[17] T. Win, H. Tianfield and Q. Mair, “Virtualization security combining mandatory access control and virtual machine introspection,” in *Proc. of IEEE/ACM International Conf. on Utility and Cloud Computing (UCC)*, pp. 1004-1009, December 8-11, 2014. Article (CrossRef Link)

[18] T. Zhang and R. Lee, “Monitoring and attestation of virtual machine security health in cloud computing,” *IEEE Micro*, vol 36, no. 5, pp. 28-37, 2016. Article (CrossRef Link)

[19] I. Studnia, E. Alata, Y. Deswarte, M. Kaâniche and V. Nicomette, “Survey of security problems in cloud computing virtual machines,” in *Proc. of Computer and Electronics Security Applications Rendez-vous (C&ESAR 2012)*, pp. 61-74, November 20-22, 2012. Article (CrossRef Link)

[20] R. Wojtczuk, “Subverting the Xen hypervisor,” in *Proc. of Black Hat USA*, August 2-7, 2008. Article (CrossRef Link)

[21] J. Rutkowska and A. Tereshkin, “Bluepilling the xen hypervisor,” in *Proc. of Black Hat USA*, August 2-7, 2008. Article (CrossRef Link)

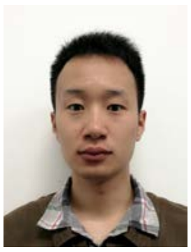
[22] C. Chen, M. Wu, B. He, X, Zheng, C. Hsing and H. Sun, “A methodology for hook-based kernel level rootkits,” in *Proc. of International Conf. on Information Security Practice and Experience (ISPEC)*, pp. 119-128, May 5-8, 2014. Article (CrossRef Link)

[23] S. Kim, J. Park, K. Lee, I. You and K. Yim, “A brief survey on rootkit techniques in malicious codes,” *Journal of Internet Services and Information Security*, vol 3, no. 4, pp. 134-137, 2012. Article (CrossRef Link)
[24] R. Lou, Y. Guo and Y. Song, “Research on trusted boot technology based on collaborative virtualization system,” *Application Research of Computers*, vol 31, no. 10, pp. 3125-3130, 2014. Article (CrossRef Link)
[25] K. Wang, Z. Li, F. Huang and F. Yan, “HyperSpector: VMM dynamic trusted monitor based on UEFI,” *Chinese Journal of Network and Information Security*, vol 2, no. 12, pp. 47-55, 2016. Article (CrossRef Link)
[26] J. Yu, P. Zhou, Y. Wu and C. Zhao, “Virtual machine replay update: improved implementation for modern hardware architecture,” in *Proc. of International Conf. on Software Security and Reliability Companion (SERE-C)*, pp. 1-6, June 20-22, 2012. Article (CrossRef Link)
[27] H. Patel, Y. Patel and H. Trivedi, “Auditing and monitoring of virtual machine instances of cloud,” *International Journal for Scientific Research & Development (IJSRD)*, vol 1, no. 2, pp. 338-341, 2013. Article (CrossRef Link)
[28] S. Oikawa and J. Kawasaki, “Simultaneous virtual-machine logging and replay,” *Simultaneous Virtual-Machine Logging and Replay*, vol 6, no. 4, pp. 1128-1138, 2011. Article (CrossRef Link)
[29] L. Catuogno, A. Castiglione and F. Palmieri, “A honeypot system with honeyword-driven fake interactive sessions,” in *Proc. of IEEE International Conf. on High Performance Computing & Simulation (HPCS)*, pp. 187-194, July 20-24, 2015. Article (CrossRef Link)
[30] N. Al-Dabagh and M. Fakhri, “Monitoring and analyzing system activities using high interaction honeypot,” *International Journal of Computer Networks and Communications Security*, vol 2, no. 1, pp. 39-45, 2014. Article (CrossRef Link)
[31] R. Tiwari and A. Jain, “Design and analysis of distributed honeypot system,” *International Journal of Computer Applications*, vol 55, no.13, pp. 20-23, 2012. Article (CrossRef Link)
[32] P. Pisarčík and P. Sokol, “Framework for distributed virtual honeynets,” in *Proc. of ACM International Conf. on Security of Information and Networks (SIN)*, pp. 324-329, September 9-11, 2014. Article (CrossRef Link)
[33] J. Qin, B. Shi and B. Li, “NEM: A new in-vm monitoring with high efficiency and strong isolation,” in *Proc. of International Conf. on Smart Computing and Communication (SmartCom)*, pp. 396-405, December 10-12, 2017. Article (CrossRef Link)
[34] B. Dolan-Gavitt, T. Leek, M. Zhivich, J. Giffin and W. Lee, “Virtuoso: Narrowing the semantic gap in virtual machine introspection,” in *Proc. of IEEE Symposium on Security and Privacy (SP)*, pp. 297-312, May 22-25, 2011. Article (CrossRef Link)
[35] Y. Liu, Y. Xia, H. Guan, B. Zang and H. Chen, “Concurrent and consistent virtual machine introspection with hardware transactional memory,” in *Proc. of IEEE, International Symposium on High Performance Computer Architecture (HPCA)*, pp. 416-427, February 15-19, 2014. Article (CrossRef Link)
[36] R. Kemmerer, “Shared resource matrix methodology: An approach to identifying storage and timing channels,” *ACM Transactions on Computer Systems (TOCS)*, vol 1, no. 3, pp. 256-277, 1983. Article (CrossRef Link)
[37] N. Kaur and A. Bindal, “A complete dynamic malware analysis,” *International Journal of Computer Applications*, vol 135, no. 4, pp. 20-25, 2016. Article (CrossRef Link)

Rui Lou was born in 1989. He received the M.S. degree in Computer Science and Technology from Information Engineering University in 2014, and B.A. degree from Harbin Institute of Technology in 2011. He is currently a Ph.D. Candidate in Computer Science and Technology in State Key Laboratory of Mathematical Engineering and Advanced Computing. His research interests include computer architecture, system virtualization, and computer security.

Liehui Jiang was born in 1967. He is currently a professor and Ph.D. Supervisor with the State Key Laboratory of Mathematic Engineering and Advanced Computing, Zhengzhou, China. His main research interests include computer architecture, reverse engineering and security. He has published over 100 refereed papers. He is a senior member of China Computer Federation.

Rui Chang was born in 1981. She received the Ph.D. degree from Information Engineering University in 2017, the M.S. degree from Wuhan University of Technology in 2007, and B.A. degree from Zhengzhou University in 2003. She is an associate Professor now in State Key Laboratory of Mathematical Engineering and Advanced Computing. Her research interests include computer architecture, embedded system security, access control, and formal method.



Yisen Wang was born in 1990. He received the M.S. degree in Computer Science and Technology from Information Engineering University in 2015, and B.A. degree from Tianjin University in 2012. Now he is a Ph.D. Candidate in Computer Science and Technology in State Key Laboratory of Mathematical Engineering and Advanced Computing. His research interests include computer architecture, internet of things security, and deep learning.