future internet

MDPI

Article

An Access Control Model for Preventing Virtual Machine Escape Attack

Jiang Wu *, Zhou Lei, Shengbo Chen and Wenfeng Shen

School of Computer Engineering and Science, Shanghai University, Shanghai 200444, China; leiz@shu.edu.cn (Z.L.); schen@shu.edu.cn (S.C.); wfShen@mail.shu.edu.cn (W.S.)

* Correspondence: wj1994@i.shu.edu.cn; Tel: +86-185-0174-6348

Academic Editor: Luis Javier Garcia Villalba

Received: 26 March 2017; Accepted: 23 May 2017; Published: 2 June 2017

Abstract: With the rapid development of Internet, the traditional computing environment is making a big migration to the cloud-computing environment. However, cloud computing introduces a set of new security problems. Aiming at the virtual machine (VM) escape attack, we study the traditional attack model and attack scenarios in the cloud-computing environment. In addition, we propose an access control model that can prevent virtual machine escape (PVME) by adapting the BLP (Bell-La Padula) model (an access control model developed by D. Bell and J. LaPadula). Finally, the PVME model has been implemented on full virtualization architecture. The experimental results show that the PVME module can effectively prevent virtual machine escape while only incurring 4% to 8% time overhead.

Keywords: virtual security; virtual machine escape; access control; BLP model; PVME model

1. Introduction

Cloud computing treats IT resources, data and applications as a service provided to the user through network, which is a change in the way of how data is modeled. Currently, the world's leading IT companies have developed and published their own cloud strategies, such as Google, Amazon, and IBM. While cloud computing is developing rapidly, more and more new technologies are emerging to protect the flexibility and scalability of the cloud-computing platform. However, security concerns have become the "stumbling block" that may jeopardize its further development and wider acceptance. Gartner surveyed 500 executives and IT managers in 17 countries. The results showed that the vast majority of users still preferred to use traditional systems rather than cloud computing systems. Even though cloud computing is envisioned as a promising service platform for the Next Generation Internet [1,2], security is one of the biggest obstacles that hamper its widespread adoption [3].

The security risks in cloud may differ from the risks of conventional IT infrastructure, either in nature or intensity or both. Resource pooling allows the use of same pool by multiple users through multi-tenancy and virtualization technologies, but in this way it also causes certain risks in the system. In paper [4,5], they survey the factors affecting cloud computing adoption and vulnerabilities and identify relevant solution directives to strengthen security and privacy in the cloud environment. In addition, a virtualized environment introduces its own set of risks and vulnerabilities that includes malicious cooperation between virtual machine (VM) and VM escape. The virtual machine escape attack was formally introduced by Grobauer et al. [6] in 2009, where virtual machine escape attack is defined as the attacker exploiting the vulnerabilities of Hypervisorto attack other virtual machines and obtain their authorities or resources. A typical scenario of virtual machine escape is as follows. A vulnerability (CVE-2008-0923) in VMware discovered by Core Security Technologies made VM escape possible on VMware Workstation 6.0.2 and 5.5.4 [7–9]. The vulnerability was found in VMware's

shared folders mechanism, where the Guestsystem can read and write any folder of the Host system including the system folder and other security-sensitive folders. Exploitation of this vulnerability allows attackers to compromise the underlying Host system.VM escape attacks are conducted at the IaaS (Infrastructure as a Service) layer, the SaaS (Software-as-a-Service) applications are built and deployed over the PaaS (Platform-as-a-Service) and the PaaS is dependent on the underlying IaaS, as a result, successful VM escape attacks are considered to be catastrophically.

As above, the threat of virtual machine escape in cloud computing needs to be solved urgently. It is of theoretical and practical significance to study how to improve the security of cloud computing and to prevent malicious attacks. In this paper, we firstly analyze the operation of the virtual machine and the Hypervisor in the virtualization architecture and explain the permission state transitions and key steps of the virtual machine escape attack. Afterwards, according to the proposed escape attack model, an access control model to prevent virtual machine escape attack (PVME) based on BLP is proposed. The PVME model can manage the system call between the virtual machine and the Hypervisor in the virtualization platform, curb the virtual machine's illegal state transition effectively, and achieve the purpose of preventing the virtual machine escape attack.

The rest of this paper is organized as follows: Section 2 describes several studies closely related to our research. Section 3 introduces some background knowledge, including Intel's hardware-assisted Virtualization Technology and two common attacking ways of VM escape. Sections 4 and 5 describe the design and implementation of our proposed model in detail. Section 6 discusses the events about the experiments. Section 7 concludes this paper.

2. Related Work

Security and privacy issues in cloud computing have received extensive attention recently, and many researchers have studied security issues in the virtualization layer [2,5]. A number of topics relate to virtualization security issues, including virtual machine image management, virtual machine monitoring, network virtualization [10], mobility and malware [11]. Among them, the virtual machine monitor (VMM) is a core issue in virtualization security. As we all know, the VMM is a software component that regulates all the virtual machines and their connection with the hardware. The core responsibility of the VMM is the management and isolation of each running VM and is also responsible for the creation and management of each virtual resource. The interconnection complexities and more entry points in the VMM can promote a large number of attack vectors.

The paper [12] discusses the Hypervisor vulnerability, along with breaking the security of the Xen and KVM (Kernel-based Virtual Machine). Authors discussed secure system isolation and presented issues that arise from strong virtualization and from weak implementation of core virtualization, but the necessary associated test procedures are not specifically implemented. Tariqul Islam et al. [13] mentioned that an access control method mainly consists of Access Control Agent (ACA) residing in the VMM and Access Control Enforcer (ACE) residing in Admin VM, which is in charge of managing the whole virtual system cooperating with the VMM. However, they only used Admin VM as an agent to conduct security policy. Liu et al. [14] proposed CATalyst, a pseudo-locking mechanism that uses CAT to partition the LLC (Last-level Cache) into hybrid hardware-software managed cache. They implemented a proof-of-concept system using Xen and Linux running on a server with Intel processors, and show that LLC side-channel attacks can be defeated. However, the Xen platform is a para-virtualized environment and platform-specific, which also makes the scope of the attack limited. Han et al. [15] proposed a prototype of such a secure policy, called the previous-selected-server-first policy (PSSF), and they defined secure metrics that measure the safety of a VM allocation policy, in terms of its ability to defend against co-resident attacks. The secure policy (PSSF) not only significantly decreases the probability of attacks co-locating with their targets, but also satisfies the constraints in workload balance and power consumption. However, once attacks co-locate with target VMs, it has no effective way to deal with attacks, and cannot solve the attacks fundamentally.

To prevent virtual machine escape attacks, some researchers proposed to restrict access to the virtual machine's resources through accessing control policies. WK Sze et al. [16] proposed a new system, called SOS, for hardening Open Stack. SOS consists of a framework that can enforce a wide range of security policies and limit trust on compute nodes. They applied mandatory access control (MAC) and capabilities to confine interactions among different components, and effective confinement policies are generated automatically. However, this method is not suitable for immediate deployment, due to the required modifications to current cloud platforms. Liu et al. [17] implemented a MAC framework application to multi-level security (MLS) in Xen on the basis of a Virt-BLP model. This model can guarantee the security of communication between VMs, but it does not take into account the risk of interaction between the virtual machines and the Hypervisor. Zhu et al. [18] proposed a multilevel security access control model V-MLR based on the mandatory access control, which not only provided secure communication mechanism for virtual machine monitor (VMM) and VMs, but also updated the borrowed information in VMM synchronously when it was changed in VMs. However, V-MLR is based on the Xen virtual machine system. Hai et al. [19] proposed a multilevel security model based on BLP model. The current security level in this model can be dynamically changed when users read sensitive data, and can guarantee that users cannot leak sensitive data after they read, but results aren't presented clearly.

The BLP model is designed for the traditional system rather than virtualization system. Although both the PVME model and virt-BLP model are based on the BLP model, they have different points. For instances, (1) we use the PVME model in full virtualized environment and mainly consider the security issues of the communications between Hypervisorand the VMs. The virt-BLP model is applied in a Xen virtual machine system, which is a para-virtualized environment and it mainly considers the security of the communications between VMs; (2) the definitions of all model elements between PVME model and virt-BLP model are different. For example, the subject or object in virt-BLP can only be VMs. However, in order to enhance the security of fully virtualized environment, the subject can only be Hypervisor or QEMU (Quick Emulator) in PVME model, and the object can be Hypervisor, GuestOS, QEMU and VMs, details in Section 4.3; (3)the structures between PVME model and virt-BLP model are totally different. We set two running modes: learning mode and enforce mode. Meanwhile we add hooking function and monitor model, and the details are in Section 5.

3. VM Escape Attack

3.1. VM Escape Attack Analysis

Intel's hardware-assisted Virtualization Technology (VT) is the first hardware-assisted virtualization solution in x86 platform. There are four different x86 processor priorities, from Ring 0 to Ring 3. Ring 0 has the highest priority and is used for the operating system kernel. Meanwhile Rings 1 and 2 are used for operating system services. Ring 3 is used for applications and has the lowest priority.

In a virtualized environment, the system kernel must run in Ring 0, while the Hypervisor and Guest OS cannot run in Ring 0. Otherwise, the system will not be stable and capable of managing all VMs effectively. Thus, the challenge is how to run a GuestOS outside Ring 0 without the auxiliary processor virtualization.

The current solution is to use Ring Deprivileging (privilege level decrease), and it has two options: GuestOS running on Ring 1 (0/1/3 model), or Ring 3 (0/3/3 model). Obviously, no matter what kind of model is used, GuestOS can never run at Ring 0.

As shown in Figure 1, the Hypervisorruns at Ring 0 of the root mode. The virtual machine emulator and host applications run on Ring 3 of the root mode. However, the kernel of the virtual machine runs at Ring 0 of non-root mode, while virtual machine applications are running at Ring 3 of non-root mode.

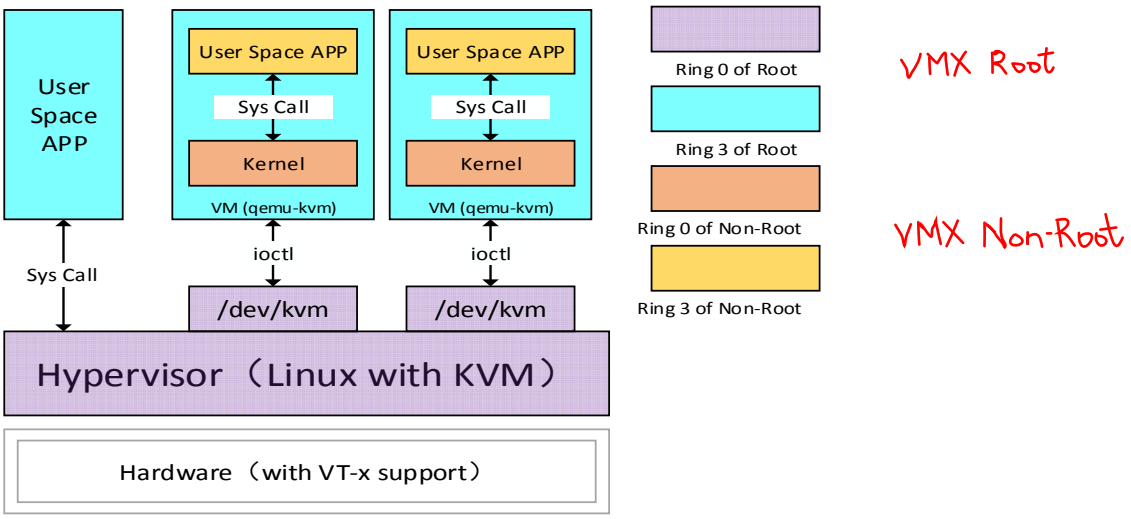

Figure 1. Hardware-assisted full virtualization.

In the virtual machine escape attack, the attacker utilizes malicious applications to obtain the highest privilege of the virtual machines, from Ring 3 of the non-root mode privilege to Ring 0 of the non-root mode privilege, and then uses the virtual machine to execute all operations given by the Hypervisor.

The interactions between the virtual machine simulator and the Hypervisorare I/O controls analog commands. Thus, in this case, the attacker can simulate the pseudo I/O operations to gain Ring 3 of the root mode privilege. Then the attacker could exploit the Hypervisorloopholes or inject code into the Hypervisor. Since the attacker has gained Ring 0 of the root mode privilege, the Hypervisor and the host operating system are in a non-secure state. Therefore the data and all the running states of the virtual machines of the host machine could be attacked or tampered with.

3.2. VM Escape Attack Model

For virtual machine escape attack, we assume that cloud service providers and infrastructure are trusted. Next we assume that a malicious user can run and control a running virtual machine. Due to the fact that cloud service providers provide services by multiplexing physical infrastructure, we also assume that the attacker's virtual machines are running on the same physical machine as a potential threat VM. So attackers can manipulate the shared physical resources, i.e., CPU cache, shared file systems, network intrusion message cues etc. to compromise the Hypervisor or host machine. We discuss the following two major attack models:

Figure 2 Virtual machine (VM) escape attack model. VMs attack Hypervisor: As shown in (**a**), an attacker uses such analog commands as ioctl and virtio to conduct overflow attacks on physical resources including CPU, memory, disk, and others. VMs attack the hosts as shown in (**b**), the attackers utilize direct communications with Host OS to tamper privileged instructions and resources of the host.

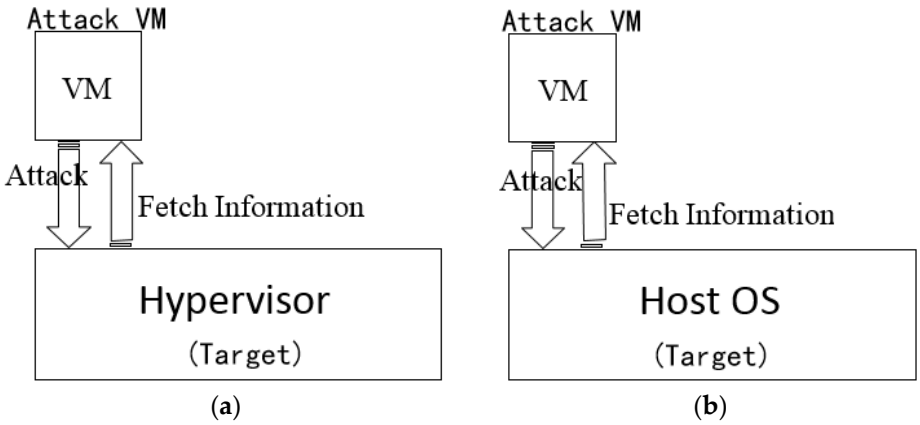

Figure 2. VM escape attack model.

In a typical case, VM escape attack includes the following steps:

1. Placement. Placement refers to the activity that malicious attackers place their malicious virtual machines on the same physical machine where the target Hypervisor or hosts run. This process is a probability event and relatively complex, which needs to use a lot of co-residency detection methods.
2. Extracting information. Once the placement is successful, the attacker can launch attacks to get key permissions of the Hypervisor or hosts, and then extract information from the other virtual machines on the host and Hypervisor.

Three challenges need to be dealt with if one wants to conduct a successful virtual machineescape attack in the cloud environment:

1. The attacker cannot determine whether the malicious virtual machine is co-resident with the Hypervisor or host that they want to attack;
2. Are there security vulnerabilities that the attacker has previously known regarding the hosts?;
3. Can the attacker know how to obtain critical information through the escape attack once they are deployed successfully?

These are the virtual machine escape attack models that we mainly consider. In the experimental section, we use a typical KVM vulnerability to simulate our attack model, and the attack procedure conforms to the proposed attack model (details in Section 6.3). Next we will verify the security of PVME model from the theoretical point, and prove the security through the real experiment in the end.

4. Access Control Model for Preventing VM Escape Attack

4.1. Access Control

Access control is a mechanism that explicitly allows or restricts access to a resource and the scope of accessibility. It is a preventive measure against unauthorized resource usage by preventing intrusion of unauthorized users or careless operations of a legitimate user [20].

There are generally three entities involved in the access control model: (1) subject: an active entity performing access operations, which is usually an application or a user process; (2) object: an object to be accessed, e.g., the data and information to be stored, resources to be accessed (i.e., the file, system or a variety of network equipment, facilities); (3) security access policy: a set of rules that determines whether a subject has permission to access an object.

Three common access control models include DAC (Discretionary Access Control), MAC (Mandatory Access Control) and RBAC (role-based access control) [21]. In addition, there are other control strategies, i.e., OBAC (Object Based Access Control) and TBAC (Task Based Access Control).

4.2. BLP Model

The BLP model was developed by D. Bell and J. LaPadula [22] in 1976. It is designed to be a military security policy that controls access to classified information by developing a set of access rules and operating authorities. From the view point of "access control", the BLP model does not only allow the subject to access to the object efficiently, but also to ensure the security of the system.

In essence, the BLP model is the first mathematical model to formalize security policies, which uses the state variables to represent the security state of the system and uses the state transition rules to describe the change processing of the system. The BLP model formalizes the definition of the system state and the transition rules between states, defines the concept of system security and sets out a set of security axioms to constrain and restrict the state and state transition rules of the system. A system is secure if its initial state is safe and the set of rules through which security requirements are maintained.

4.3. PVME Model

The BLP model is originally designed for a traditional operating system environment. Taking into account the characteristics of virtualized environments and their differences from the traditional operating system environment, the PVME model applies the basic principles of the BLP model and adapts security axioms and state transition rules of the BLP model to accommodate for the virtual machine escape (mainly Hypervisor communication) scenario.

4.3.1. Model Elements

The PVME model uses the same symbols of the BLP model. The basic elements include: subject, object, access attribute set, main object security level, access control matrix, object level and system status. The following is a detailed description of the basic elements of PVME model.

1. Subject: the subject represents the active side that issues access operations and access requests. It usually refers the users or processes that can make the information flow. The capital S represents the subject set and the lowercase s represents a single subject, namely $S = \{s_1, s_2, \ldots, s_n\}$.
2. Object: the object represents the side that can be accessed, generally documents, procedures or equipment that can be called. The capital O represents the subject set and the lowercase o represents a single subject, namely $O = \{o_1, o_2, \ldots, o_n\}$.
3. Access Attribute Set: access set $A = \{r, a, w, e^*, c^*\}$. In our research environment, the communicating parties involved Hypervisor, VM, QEMU and Guest OS. However the subject can only be Hypervisor or QEMU and the object can be Hypervisor, Guest OS, VM or QEMU. When a virtual machine communicates with the Hypervisor, in addition to traditional access modes (read-only (r), write-only (a), write-read (w)), we redefine the implementation of access mode (e^*), and add a new access mode (c^*). Since the execution (e) access in the BLP model is completely redefined in the PVME model, we use e^* to make that distinction. The execution (e^*) and control (c^*) modes are the only two operations that exist when theHypervisor communicates with virtual machines. Therefore, the access attribute set $A = \{r, a, w, e^*, c^*\}$.
4. Access Control Matrix: the element m_{ij} of access control matrix M represents the subject S_i owns the access to the object O_j in current state.
5. Security Level: the security level contains security classification and security category. $C = \{C_1, C_2, \ldots, C_n\}$ denotes the security classification set where $C_1 > C_2 > \ldots > C_n$ and $K = \{K_1, K_2, \ldots, K_m\}$ represents the security category set whose element $K_i (1 \leq i \leq m)$ is a specific access property. Security level set $L = \{L_1, L_2, \ldots, L_p\}$ is a partial ordered set, each item in the set $L_i = (C_i, K_i)$ represents a security level, which $C_i \in C$, $K_i \subseteq K$, $1 \leq i \leq p$. If and only if $C_i \geq C_j \cup K_i \supseteq K_j$ then we get $L_i \geq L_j$. Each element in security level vector set $F\ (F \subseteq L^s \times L^o \times L^s)$ is $f = (f_s, f_o, f_c)$ where f_s is the security level function of the object, f_o is the security level function of the object and f_c is the current security level function of the subject.

6. Object Hierarchy: object level set H indicates the subordinate relationship between subject and object. In the object hierarchy, a node mostly has one and only one parent without a hierarchy ring. The symbol $O_j \in H(O)$ represents O_j is the leaf node and O is the parent node in the hierarchy.
7. System Status: system Status collection is a four-tuple $V = \{B \times M \times F \times H\}$, every single element quadruple $v = \{b \times M \times f \times H\}$ represents a system status, in which B, M, F, H are the current accessing set, access control matrix, security level vector set and the hierarchy of the objects, respectively. The symbol $B = P(S \times O \times A)$, and its element b $(b \subseteq S \times O \times A)$ is the current access set, which records what subject can access what objects in which access property.
8. Request Element Set: request element set $RA = \{g, r\}$, where the g denotes get or give the requests and the r denotes release or rescind the requests. The request set $R = \left\{R^{(1)}, R^{(2)}, R^{(3)}\right\}$, whose function is as shown Table 1.

Table 1. The function of request set.

The Element of Request Set	Function
$R^{(1)} = RA \times S \times O \times A$	The subject requires or releases some access to the object.
$R^{(2)} = RA \times S \times O \times L$	The subject requires creating the object or changes its security level.
$R^{(3)} = S \times O$	The subject requires deletingtheobject.

The related elements and corresponding access processes of the PVME model are based on a full virtualization platform. From bottom to top in Figure 3: host OS represents the host operating system; Hypervisor represents the corresponding virtual machine monitor. Since the current mainstream virtual machine simulator is mostly based on QEMU, we use QEMU to represent the virtual machine simulator of the whole virtual architecture. Guest OS stands for the virtual machine operating system. In this architecture, VM is mainly used to express the whole QEMU and Guest OS.

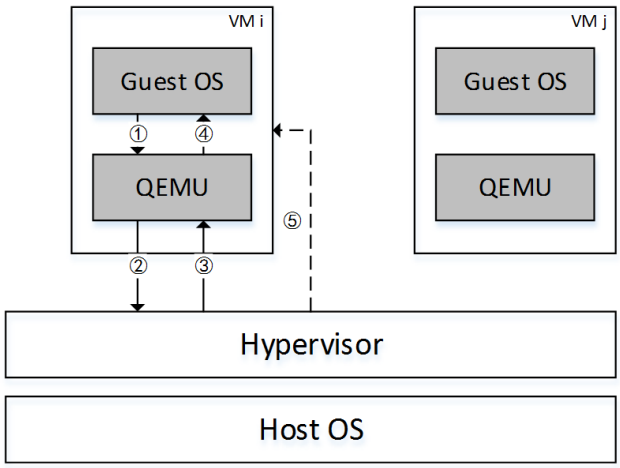

Figure 3. VM access mechanism under full virtualization architecture.

The PVME model is designed to prevent virtual machine escape attacks in the virtualization system. As shown in Figure 3, the hardware command issued by the Guest OS will be simulated by QEMU, and then was sent to Hypervisor as process ① and process ② show, and process ③, ④ represent the response to Guest OS's hardware request. For virtual machine escape, we mainly prevent the attack from Guest OS to Hypervisor or hosts, without considering the impact of Hypervisor on the Guest OS. Therefore the PVME model combines process ③, ④ into process ⑤, which represents

communication between VMs and Hypervisor. The PVME model mainly considers processes ①, ②, ⑤ in the full virtualized architecture.

The set $S^*(S^* \subseteq S)$ includes subjects that meet with the * -property of BLP model, $S_T = S - S^*$ denotes the set of trusted subjects. The Hypervisor is used for managing other virtual machines on the host and has the absolute authority above the virtual machines. When Hypervisor is a subject, we define it as a trusted one and it belongs to the trusted set S_T. In fact, Hypervisor is the only trusted subject. When Hypervisor is an object, it serves as the root node of the object set, namely O_R. In this case, when the Guest virtual machines are objects, in the object hierarchy, they are child nodes of Hypervisor (the root), and are located in the same object hierarchy. Obviously, in the PVME model the object hierarchy maintains the two significant properties of the BLP model, i.e., in the object hierarchy, a node at most has one parent node, and there is no cycle in the object hierarchy.

4.3.2. Security Axioms

All security axioms of PVME model are named with PVME- as a prefix.

Axiom 1 (PVME-*-property). *Set S' is a subset of S in which every state $v = (b \times M \times f \times H)$ meets PVME-*-property, iff:*

$$s \in S' \Rightarrow \begin{cases} (O \in b(S : a)) \Rightarrow (f_o(O) \geq f_s(S)) \\ (O \in b(S : w, e^*, c^*)) \Rightarrow (f_o(O) = f_s(S)) \\ (O \in b(S : r)) \Rightarrow (f_o(O) \leq f_s(S)) \end{cases}$$

Axiom 2 (PVME-ds-property). *We redefine control (c^*) attribute and add execution (e^*) attribute in the access attributes. That is expressed as: a state $v = (b \times M \times f \times H)$ satisfies PVME-ds-property (PVME-customized security property) if and only if: $\forall(s, o, x) \in b \&\& x \in M_{ij}$, where x is one of the five access properties (read-only (r), write-only (a), write-read (w), execute (e*) and control (c*)).*

Axiom 3 (Delete PVME-ss-property). *Similar to the ss-property of BLP model, we assume that there is a property called PVME-ss-property in the PVME model. However, Hypervisor should communicate with Guest VMs when managing them and it has full access (i.e., read-only (r), write-only (a), write-read (w), execute (e*) and control (c*)) to Guest VMs. Thus, Hypervisor belongs to a trusted body set S_T when it is a subject. Clearly S_T does not satisfy the PVME-ss-property. For all of the Guest virtual machines, if they meet PVME-*-property, it is easy to prove that they also meet PVME-ss-property. Therefore, due to the presence of Hypervisor, we must remove the PVME-ss-property.*

The system security needs to be changed appropriately because of the absence of PVME-ss-property in the PVME model. That is, a security state v should satisfy PVME-ds-property and PVME-*-property about S^*, while a security state transition rule must maintain PVME-*-property and PVME-ds-property. Other concepts in the PVME model such as security state sequence, system security and safety systems are consistent with the BLP model.

4.3.3. State Transition Rules

The PVME model includes 9 state transition rules, denoted $\mathrm{PVME-R_i}$, $1 \leq \mathrm{i} \leq 9$. The input is a (request, state) pair and correspondingly the output is a (result, state) pair, which is similar to the BLP model. The following is a brief introduction of some symbols used in the description of state transition rules.

In the input pairs, the form of the request is different in different rules, but the expression of the state is always $v = (b \times M \times f \times H)$. Therefore we only need to concern the request section of the rule's domain (definitional domain). Accordingly, the rule's domains are recorded as domain $PVME - R_i$, $1 \leq i \leq 9$. In the output pairs, result is defined as the set $D = \{yes, no, ?\}$, where *yes*

means the request will be permitted and executed; *no* means the request will not be permitted and executed; *?* means that the request is beyond the scope of the rule, that is, the request does not belong to the domain request.

The main differences between the PVME model and the BLP model on transition rules are the following:

1. In the PVME model when the Hypervisor managers the virtual machines, in this pair the subject can only be theHypervisor. Besides, the PVME model does notconsider the communications between VMs. As a result, the PVME model removes the rules corresponding to the rule 6 and 7 in BLP model.
2. The PVME model redefines the execution (*e**) access property and adds a new rule (releasing execution (*e**) access property).
3. The Hypervisor has the control (*c**) access property when it manages VMs. Thus, we add two rules (acquiring and releasing control (*c**) access property).
4. In the PVME model, the virtual machines have the same security level, no matter whether they are subjects or objects. Thus, we remove rule 10 of the BLP model and only keep the rule that changes objects' security level.

The following is a detailed introduction of the composition of state transition rules in the PVME model, which is mainly divided into two cases:

1. If the subject is Hypervisor and the object is QEMU or the subject is QEMU and the object is Guest OS, we denote the subject as S_i (if the subject is Hypervisor, $i = 1$; if the subject is QEMU (assuming that there are n VMs), $1 \leq i \leq$ n) and designate the object as $O_j (1 \leq j \leq n)$, at this time the subject only S_i has three accesses (read-only (*r*), write-only (*a*), write-read (*w*)) to the object O_j, which is mapped to twostate transition rules.
2. If the subject is theHypervisor and the objects are VMs, we denote the subject as $S_{Ti} (i = 1)$ and denote the object as $O_j (1 \leq j \leq n)$. As theHypervisor has absolute control over VMs, the subject S_{Ti} can get access to subject O_j in five ways (read-only (*r*), write-only (*a*), write-read (*w*), execute (*e**) and control (*c**)).In this case, on one hand the paper integrates the three access methods (the Hypervisor gets three types of access to the VMs, i.e., read-only (*r*), write-only (*a*), write-read (*w*)) into the first case, on the other the transition rules for the Hypervisor to get execute (*e**) and control (*c**) access to the VMs will be individually designed, which can totally map into four state transition rules. Rules 7–8 are used for subject Hypervisor to create and delete object VMs, and rule 9 can be used to change the security level of the object VMs.

Through the above analysis, in the PVME model, the corresponding set of requests is modified to Table 1 that differs from the request set of BLP model. The following is a detailed explanation of the 9 rules of the PVME model.

Rule 1 (PVME-R_1 (get-read/append/write)). *If the subject Hypervisor or QEMU. S_i (if the subject is the Hypervisor, $i = 1$; if the subject is QEMU S_i, $1 \leq i \leq n$) needs to obtain the x (x is one of the read-only (r), write-only(a), write-read(w)) access to O_j (the object is QEMU or Guest OS) or S_{Ti} (the subject is theHypervisor) needs to get x access to O_j (the object is Guest OS), the definition is domain $(PVME - R_1) = \left\{ R_k \middle| (g, S_i, O_j, x) \in R^{(1)} \right\}$, $x \in \{a, r, w\}$. The rules is follows:*

$$PVME - R_1(R_k, v) = \begin{cases} (?, v) & if\ R_k \notin domain(PVME - R_1) \\ (yes,\ (b \cup \{(S_i,\ O_j,\ x)\}, M, f, H)) & if[R_k \in domain(PVME - R_1)] \\ & and\,[x \in M_{ij}] \\ & and\,[f_s(S_i) \geq \mathrm{f_o}(O_j)\ or\ S_i \in S_T] \\ (no,\ v) & otherwise \end{cases}$$

If the decision is yes, adding x the way the subject S_i getting access to the object O_j to the current accessing set.

Rule 2 (PVME-R_2 (released-read/append/write)). *If the subject Hypervisor or QEMU. S_i (if the subject is the Hypervisor, i = 1; if the subject QEMU (assuming that there are n VMs), $1 \leq i \leq n$) wants release the x (x is one of the read-only (r), write-only (a), write-read (w)) access to the O_j (the object is QEMU or Guest OS) or the S_{Ti} (the subject is theHypervisor) needs to release the x access to O_j (the object is VM), the definition is domain $(PVME - R_2) = \left\{ R_k \middle| (r, S_i, O_j, x) \in R^{(1)} \right\}$, $x \in \{r, a, w\}$, the rule is as follows:*

$$PVME - R_2(R_k, v) = \begin{cases} (yes, (b \cup \{(S_i, O_j, x)\}, M, f, H)) & if\ R_k \in domain(PVME - R_2) \\ & and\ x \in \{r, a, w\} \\ (?, v) & otherwise \end{cases}$$

If the decision is yes, removing x, where x is read-only (r), write-only (a), or write-read (w), from the set of accesses allowed for subject S_i to access object O_j.

Rule 3 (PVME-R_3 (get-execute)). *The S_{Ti} (the subject is theHypervisor, i=1) needs to obtain the execute (e*) access to O_j (the objects are VMs, $1 \leq i \leq n$), the definition is domain $(PVME - R_3) = \left\{ R_k \middle| (g, S_i, O_j, e^*) \in R^{(1)} \right\}$, and the rule is as follows:*

$$PVME - R_3(R_k, v) = \begin{cases} (?, v) & if\ R_k \notin domain(PVME - R_3) \\ & if\ [R_k \in domain(PVME - R_3)] \\ (yes, (b \cup \{(S_i, O_j, e^*)\}, M, f, H)) & and [e^* \in M_{ij}] \\ & and [f_s(S_i) \geq f_o(O_j)\ or\ S_i \in S_T] \\ (no, v) & otherwise \end{cases}$$

If the decision is yes, then adding execute (e) into the set of accesses allowed for subject S_i to access object O_j.*

Rule 4 (PVME-R_4 (release-execute)). *The S_{Ti} (the subject is the Hypervisor, i=1) needs to release the execute (e*) access to O_j (the objects are VMs, $1 \leq i \leq n$), the definition is domain $(PVME - R_4) = \left\{ R_k \middle| (r, S_i, O_j, e^*) \in R^{(1)} \right\}$, and the rule is as follows:*

$$PVME - R_4(R_k, v) = \begin{cases} (yes, (b \cup \{(S_i, O_j, e^*)\}, M, f, H)) & if\ R_k \in domain(PVME - R_4) \\ (?, v) & otherwise \end{cases}$$

If the decision is yes, then removing execute (e) from the set of accesses allowed for subject S_i to access object O_j.*

Rule 5 (PVME-R_5 (get-control)). *S_{Ti} (the subject is the Hypervisor, i = 1) needs to obtain the control(c*) access to O_j (the objects are VMs, $1 \leq i \leq n$), the definition is domain $(PVME - R_5) = \left\{ R_k \middle| (g, S_i, O_j, c^*) \in R^{(1)} \right\}$, and the rule is as follows:*

$$PVME - R_5(R_k, v) = \begin{cases} (?, v) & if\ R_k \notin domain(PVME - R_5) \\ & if\ [R_k \in domain(PVME - R_5)] \\ (yes, (b \cup \{(S_i, O_j, c^*)\}, M, f, H)) & and [c^* \in M_{ij}] \\ & and\ [f_s(S_i) \geq f_o(O_j)\ or\ S_i \in S_T] \\ (no, v) & otherwise \end{cases}$$

If the decision is yes, then adding control (c) to the set of accesses allowed for subject S_i to access object O_j.*

Rule 6 (PVME-R_6 (release-control)). *S_{Ti} (the subject is the Hypervisor, $i = 1$) needs to release the control (c*) access to O_j (the objects are VMs, $1 \leq i \leq n$), the definition is domain $(PVME - R_6) = \left\{R_k \middle| (r, S_i, O_j, c^*) \in R^{(1)}\right\}$, and the rule is as follows:*

$$PVME - R_6(R_k, v) = \begin{cases} (yes, (b \cup \{(S_i, O_j, c^*)\}, M, f, H)) & if\ R_k \in domain(PVME - R_6) \\ (?, v) & otherwise \end{cases}$$

subtract

If the decision is yes, then removing the control (c) the way the subject S_i getting access to the object O_j from the current accessing set.*

Rule 7 (PVME-R_7 (create-vm-object)). *S_{Ti} (the subject is the Hypervisor, i=1) needs to create O_j (the objects are VMs, $1 \leq i \leq n$), the definition is domain $(PVME - R_7) = \left\{R_k \middle| (g, S_i, O_j, L_u) \in R^{(2)}\right\}$, and the rule is as follows:*

$$PVME - R_7(R_k, v) = \begin{cases} (?, v) & if\ R_k \notin domain(PVME - R_7) \\ & if [R_k \in domain(PVME - R_7)] \\ (yes, \begin{pmatrix} b, M, f \backslash f_o \leftarrow f_o \cup (O_j, L_u) \\ , H \cup (O_R, O_j) \end{pmatrix}) & and [S_i \in S_T] \\ (no\ , v) & otherwise \end{cases}$$

In PVME-R_7, the symbol $\leftarrow$ represents that the right side $f_o \cup (O_j, L_u)$ is assigned to the left side (f_o). The symbol (O_j, L_u) means $f_o(O_j) = L_u$ and the symbol (O_R, O_j) expresses $H(O_R) = O_j$. The symbol in expression $A \backslash B$ suggests that A will be modified by B. Therefore the expression $f \backslash f_o \leftarrow f_o \cup (O_j, L_u)$ of rule $PVME - R_7$ denotes changing the security level of object O_j in the security level vector f to L_u. The following rules will continue to use this expression.

If the decision is yes, the Guest VM will be created and the related elements will be added to security and object hierarchy.

Rule 8 (PVME-R_8 (delete-vm-object)). *S_{Ti} (the subject is the Hypervisor, i=1) needs to delete O_j (the objects are VMs, $1 \leq i \leq n$), the definition is domain $(PVME - R_8) = \left\{R_k \middle| (S_i, O_j) \in R^{(3)}\right\}$, and the rule is as follows:*

$$PVME - R_8(R_k, v) = \begin{cases} (?, v) & if\ R_k \notin domain(PVME - R_8) \\ & if [R_k \in domain(PVME - R_8)] \\ (yes, \begin{pmatrix} (b - acc(O_j) - oper(S_j)), \\ M \backslash \{M_{uj} \leftarrow \varnothing, M_{ju} \leftarrow \varnothing\}, f, \\ H - (O_R, O_j) \end{pmatrix}) & and [S_i \in S_T] and [S_j \notin S_T] \\ & and [O_j \neq O_R] \\ (no, v) & otherwise \end{cases}$$

The parameter n is the total number of the VMs and $1 \leq u \leq n$. Expression $acc(O_j) = (S \times \{O_j\} \times A) \cap b$ represents ~~that~~ the current access set b has ~~all accesses to O_j~~ (Guest VMs). ~~However~~, Besides expression $oper(S_j) = (\{S\} \times O_j \times A) \cap b$ indicates ~~that~~ the deleted VMs in the current access set b ~~can be visited as the objects by the subject S_j with all the access ways~~.

all the access to O_j · all the access ways · has · when it is acting as a subject S_j

If the decision is yes, the Guest VMs will be deleted and at the same time the related elements of current access set b or access matrix M or object hierarchy H will be removed as well.

Rule 9 (PVME-R_9 (change-vm-object-security-level)). *Hypervisor needs to change the security level L_u of object VM. The definition is domain* $(PVME-R_9) = \left\{ R_k \middle| (g, S_i, O_j, L_u) \in R^{(2)} \right\}$, *and the rule is as follows:*

$$PVME-R_9(R_k, v) = \begin{cases} (?, v) & \text{if } R_k \notin domain(PVME-R_9) \\ & \text{if}[R_k \in domain(PVME-R_9)] \\ (yes, \begin{pmatrix} b, M, f \backslash f_o \leftarrow f_o \cup (O_j, L_u) \\ , H \cup (O_R, O_j) \end{pmatrix}) & and[S_i \in S_T] and [O_j \neq O_R] and \\ [PVME-*9(R_k, v) = true] & \\ (no, v) & otherwise \end{cases}$$

PVME--property, in this case after the function's conversion the state v still maintains the PVME-*-property. Specifically:* $PVME-*9(R_k, v) = true \Leftrightarrow \forall S_\lambda \in S', \left[(S_\lambda, O_j, a) \in b \Longrightarrow L_u \geq f_s(S_\lambda)\right] \& \left[(S_\lambda, O_j, w) \in b \Longrightarrow L_u = f_s(S_\lambda)\right] \& \left[(S_\lambda, O_j, r) \in b \Longrightarrow L_u \leq f_s(S_\lambda)\right] \forall O_\lambda \in O, \left[(S_\lambda, O_j, a) \in b \Longrightarrow f_O(O_\lambda) \geq L_u\right] \& \left[(S_\lambda, O_j, w) \in b \Longrightarrow f_O(O_\lambda) = L_u\right] \& \left[(S_\lambda, O_j, r) \in b \Longrightarrow f_O(O_\lambda) \leq L_u\right]$.

If the decision is yes, the security level of the object VM will be changed to L_u.

For the whole system, if its original state z_o is safe and its state transitions are onthe basis of the rules $PVME-\{R_1, R_2, \ldots, R_9\}$, then we can conclude that the system maintains PVME-*-property and PVME-ds-property and is a security system.

4.4. PVME Model Design

4.4.1. Access Matrix

We put the access matrix into the Hypervisor, and store it as a binary file. For the sake of reliability, we store a backup access matrix file to the Host OS as well. The basic structure of the access matrix:

$$[SID, OID, R, A, W, E^*, C^*, Flag]$$

The symbols *SID* and *OID* represent the ID of the subject and object respectively, which are thirteen-bit binary numbers. Meanwhile, *R, A, W, E*, C** respectively means read-only, write-only, write-read, execute, and control access, which are represented as a binary number (the value of 1 means permission and 0 means refuse). Thus, we need a total of five bits. The last bit of the access matrix is the Flag bit that indicates whether the rule is valid (1 means valid and 0 donates invalid). Therefore the access matrix has in total 32 (13+13+5+1=32) binary bits, namely 4 bytes.

For example, a record of the access matrix:

$$1001\ 0110\ 1110\ 1110\ 0110\ 0100\ 1001\ 0101 \quad ()$$

It means the subject 1001011011101 has the 01010 (write-only, execute) access mentions to object 1100110010010 and the rule is valid.

4.4.2. Security Level

The security level consists of security classification and security category. There are 8 security classifications in the PVME model: security classification $C = \{C_1, C_2, \ldots, C_8\}$ where C_1 has the highest level and C_8 has the lowest level. We use 3-bit binary number to represent the security classification, i.e., C_1 is 111 and C_8 is 000. However, there are 16 security categories: $K = \{K_1, K_2, \ldots, K_{16}\}$, also known as 16 kinds of access. If $K_1 = 1 (1 \leq i \leq 16)$, the subject has K_i access to the object. Thus, a record is made up of 32 (13+3+16) binary bits, which is 4 bytes.

For example, the following is a record of security level:

$$0010\ 1100\ 1110\ 1010\ 1101\ 0000\ 0000\ 0000 \quad ()$$

It means the ID of the subject or object is 0010 1100 1110 1 and its security classification is C_6 (010), its security category is $\{K_1, K_2, K_4\}$ (1101 0000 0000 0000).

4.4.3. Current Access Matrix

The current access matrix b $(b \in S \times O \times A)$ includes the accesses that the subject id has for the object at present, which is used to determine whether the system is safe. Every subject has its own current access matrix b, $b(S) \in O \times A$ in the PVME model. The elements of the object set O are represented by OIDs and the access property set consists of r, a, w, e^*, c^*.

5. Design of the PVME Model

We will discuss the design of the PVME model that prevents the virtual machine escape attack in this section.

Figure 4 is the detailed architecture diagram of the PVME model combined with the design concepts of reference monitor. The dark module is the newly added one, including the QEMU_Compliance module and Hy_PVME model. The main task of the QEMU_Compliance module is to check the compliance of the request sent from the Guest OS. In particular, it filters the invalid or malicious requests such that only the compliant request can pass the QEMU and received by the Hypervisor.

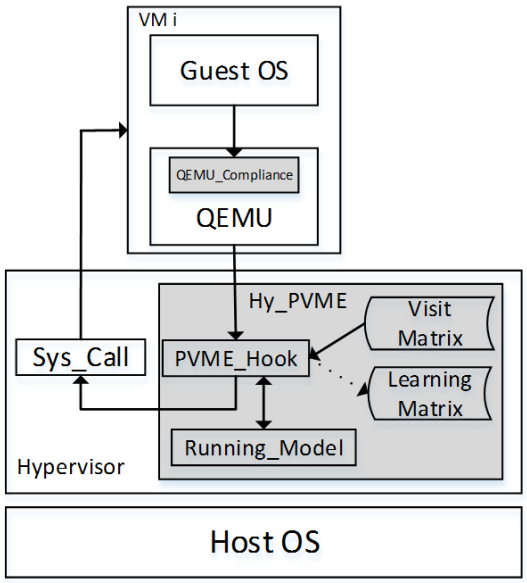

Figure 4. PVME model architecture diagram.

The Hy_PVME module includes four parts: PVME_Hook, Running_Model, Visit Matrix and Learning Matrix, which respectively stand for the hook function of the PVME model, compliance at the running mode, access rule matrix and learning matrix. The PVME model has two running modes: Learning mode and Enforce mode. The Learning mode is used in the pre-simulation environment to record all operations of the VMs in Learning Matrix. Then these records will be analyzed, deduplications combined and added to the Visit Matrix. The Enforce mode means when the PVME model is under this mode all compliant operations must comply with the rules of Visit Matrix. Adding a Learning mode can help us collect more noncompliant operations, which can make our PVME model more versatility. The PVME hook function is mainly used to receive messages from the

QEMU and send them to the PVME model to check compliance, i.e., matching them with the data of Visit Matrix. If the result satisfies the rules then the Sys_Call (system call) of the Hypervisor is returned. If not, the system will verify the running mode (through the Running_Model) of the PVME model. If the running mode is Enforce then return Error and reject the detailed request message. If the running mode is Learning then the system will return Sys_Call and record the message to the Learning Matrix.

Figure 5 shows the detailed process of the PVME model. A request by Guest OS will go through the QEMU_Compliance module and the Hy_PVME module, which will enter Error state if the rules defined in the PVME model is not met; otherwise, it will be passed to the default Sys_Call in the Hypervisor.

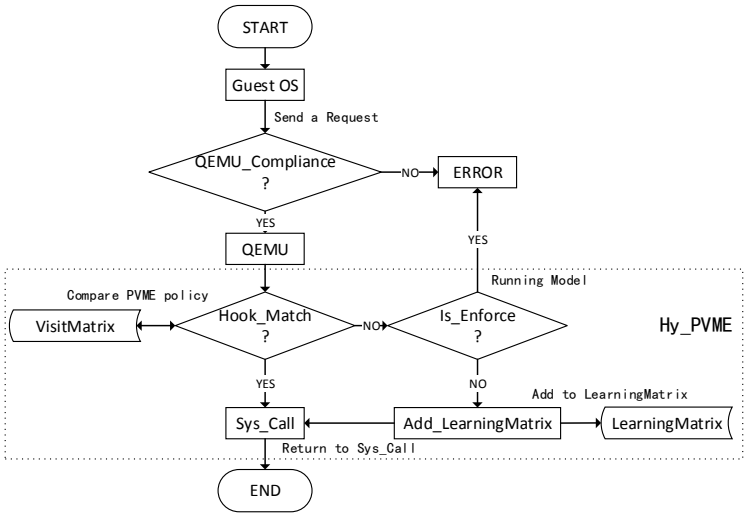

Figure 5. PVME model flow diagram.

6. Experiments

6.1. Basic Environment

This experiment is performed using a PC with the following configuration: CPU: Pentium (R) Dual-Core E5700, clocked at 3.00 GHz; Memory Capacity: 4 GB; Hard disk capacity: 500GB.

We use KVM in the experiment as a fully virtualized environment. The operating system is CentOS. Table 2 shows the basic environment configuration.

Table 2. Basic software environment.

Machine	System Version	Kernel Version	With GUI
Host machine	CentOS-6.3	2.6.32_279.e16.x86_64	Yes
Virtual machine	CentOS-mini-6.3	2.6.32_279.e16.i686	No

We choose CentOS with a graphical interface as the host operating system. This allows us to connect to the virtual machine to perform operations. However, only the basic environment of the virtual machine system is required. Thus, we choose the streamlined version of CentOS (CentOS-mini, without GUI) as the virtual machine operating system

We use the tool qemu-kvm (version of qemu is 0.12.1) to create virtual machines and use VNC (Virtual Network Computing) as the remote desktop protocol to connect to a virtual machine.

6.2. Initialization of the PVME model

The running mode of the PVME model can be the Enforce or Learning mode. We can initialize the PVME model through the Learning mode. That is, Visit Matrix and Learning Matrix are both empty at

the initial state, and Running_Mode is set to be Learning. At this time it is impossible to determine whether the operations of VMs are compliant with the rules of the PVME model in the rule bank. However, all operations in the VMs will be recorded in the Learning Matrix.

We conducted a series of normal operations on a virtual machine, including creating/deleting files, browsing the web, getting access to shared folders and so on. Figure 6 shows the data of Learning Matrix.

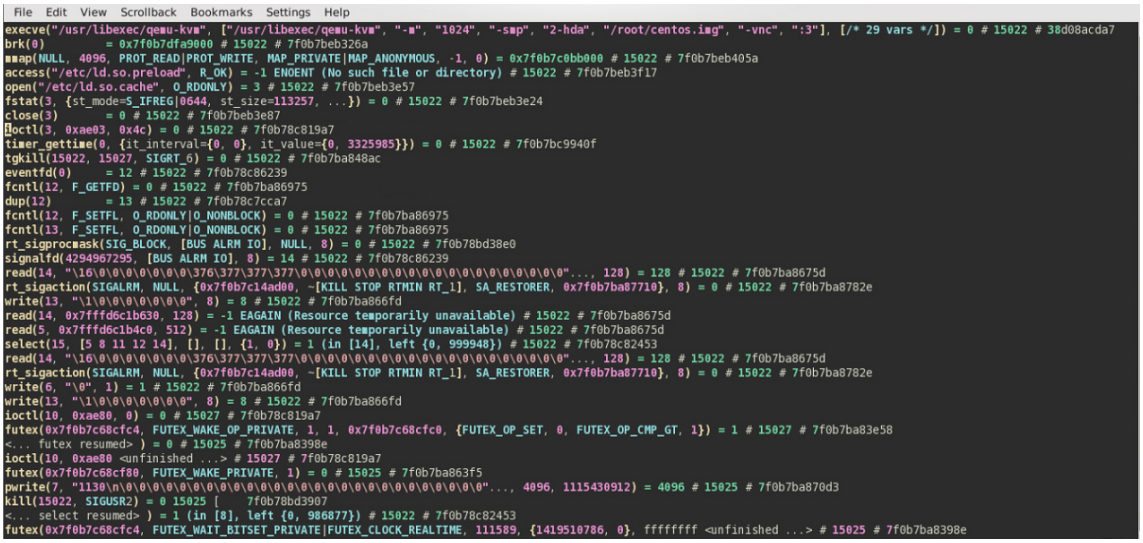

Figure 6. The data of Learning Matrix.

Then the records will be removed, the deduplications combined, analyzed and added to the Visit Matrix, as shown in Table 3.

Table 3. The data of Visit Matrix.

SID	OID	Access Property					Flag
		R	A	W	E^*	C^*	
1101 0101 0111 0	1110 1111 0000 1	1	1	1	0	0	1
1101 0101 0111 0	1111 0111 1101 0	1	1	0	0	0	1
0010 1001 0101 0	1101 0101 0111 0	0	1	0	0	0	0
0010 1001 0101 0	0010 1001 0101 0	1	1	0	1		1

The data of the Learning Matrix in Figure 6 is divided into three sections by #, i.e., write (13, \1\0\0\0\0\0\0\0, 8) = 8 #15022 #7f0b7ba866fd.

The first section is divided into two parts by '='. The first part is the function name and parameters, and the second part is the return value.

The second section is the ID of process and the last section is the entry point of the called function. This structure corresponds to the pseudo-code structure (Guest_OS_Request, SID, OID). After analyzing the data of Learning Matrix we obtain Visit Matrix, which is shown in Table 3.

The data in Table 3 are a particular example of the Visit Matrix, namely the access rule of the PVME model. For example, the first two items are the access properties a VM owned to the particular resources of the Guest machine. The latter two items are the access properties the Hypervisor possessed to the specific VM.

All subjects and objects in Visit Matrix have their own security classifications and security categories, as shown in Table 4.

Table 4. The security classification and category of object/subject.

ID	Security Classification	Security Category
0010 1001 0101 0	110	1110 1000 0000 0000
1101 0101 0111 0	011	0101 1000 0000 0000
1101 0101 0111 1	011	1101 1000 0000 0000
1110 1111 0000 1	010	0001 1000 0000 0000
1111 0111 1101 0	001	0101 0000 0000 0000

If the security classification of the subject is higher than that of the object and the security category of the object belongs to that of the subject, the subject can dominate the subject and the useful access attributes is in the Visit Matrix.

6.3. Simulation of Preventing VM Escape Attack

The PVEM model is designed to prevent virtual machine escapes in a cloud-computing environment. In the KVM environment, there is an external device vulnerability [23], which uses the "use-after-free" vulnerability to lunch Guest OS to Host OS overrun attack, and this section will demonstrate that the PVME model can achieve the goal of preventing attacks.

Article [23] describes the security hole CVE-2011-1751 in the KVM environment, where KVM cannot handle the illegal or accidental extraction of external devices, which results in a problem that the system is unable to immediately clean up the damaged states and dangerous points caused by the external devices. The attacker embeds malicious code to the external devices which include special chipsto illegally tamper with the virtual machine network stack, physical memory mapping and so on. This causes the function second_timer2 not to return to the final state and function rtc_update_second to wait indefinitely. As a result, the RTC (Real-Time Clock) of the Hypervisor is updated improperly. Refer to [23] for the details of the attack process as well as each of the functions involved in the process.

To prevent the virtual machine escape attacks caused by the RTCState attack, we add the security level as Table 5. As shown in Table 5, there are five newlyadded subjects/objects. For ease of notation, we set their ID as 1, 2, 3, 4, and 5. As the RTC is in the Hypervisor, we set its security classification as C_4. The next_second_time variable is used for scheduling rtc_update_second and ultimately realizing the normal adapt of RTC. Thus, we set its security classification as C_5; theoperation of rtc_update_second is in accordance with the return state of the virtual machine equipment and its security classification is set to C_6. Because second_timer and second_timer2 are in virtual machine devices, their security classifications are C_7 and are relatively low.

Table 5. Security level for the RTCState attack.

Subject/Object	Id	Security Classification	Security Category
RTC	0000 0000 0000 1	100	1111 0000 0000 0000
next_second_time	0000 0000 0001 0	011	1110 0000 0000 0000
rtc_update_second	0000 0000 0001 1	010	1110 0000 0000 0000
second_timer	0000 0000 0010 0	001	1010 0000 0000 0000
second_timer2	0000 0000 0010 1	001	1010 0000 0000 0000

Table 6 is an access matrix for the prevention of RTCState attacks, which can be summarized as follows: The higher security level has the read property to the lower security level, while the lower security level can read and write the higher security level.

We start the virtual machine in the running mode of PVME model (it is in the Enforce mode with loading existing and newly added access matrix) and insert external devices (using U disk as the external device) on the host machine. One U disk has the malicious code similar to that in article [23].

Then we redirect the U disk to a virtual machine, making it the dedicated device of the virtual machine (the virtual machine can read or write this U disk). At this time we have finished the Placement step of the VM escape attack model. The result of this step is shown in Figure 7.

Table 6. Access matrix for RTCState attack.

SID	OID	Access Property					Flag
		R	A	W	E^*	C^*	
0000 0000 0000 1	0000 0000 0001 0	0	0	0	0	0	1
0000 0000 0001 0	0000 0000 0000 1	1	0	1	0	0	1
0000 0000 0001 0	0000 0000 0001 1	1	0	0	0	0	1
0000 0000 0001 1	0000 0000 0001 0	1	0	1	0	0	1
0000 0000 0001 1	0000 0000 0010 0	1	0	0	0	0	1
0000 0000 0010 0	0000 0000 0001 1	1	0	1	0	0	1

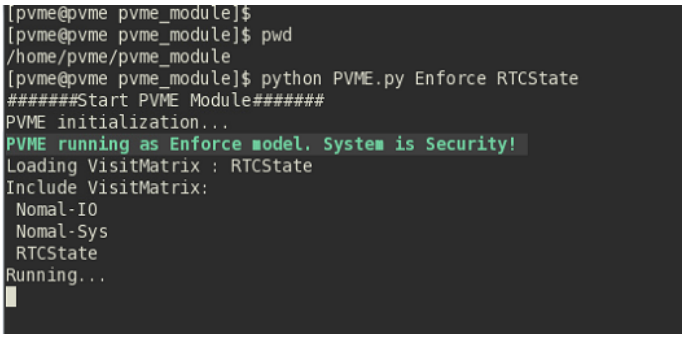

Figure 7. Load the access matrix of RTCState in PVME model.

Executing the malicious code segment of the U disk on the virtual machine after the successful placement, we illegally extract the U disk from the host machine. At this point we can simulate the attack scenarios of RTCState, making the RTC of the Hypervisor unable to achieve thenext_second_time update status. This would result in the VM successfully changing the status of the RTC at last. This completes the information extraction step of the virtual machine escape attack model. According to the above analysis, the RTCstate attack satisfies the attack model proposed in Section 3.2. Performing the above operations in the PVME model, we can get the out log as shown in Figure 8.

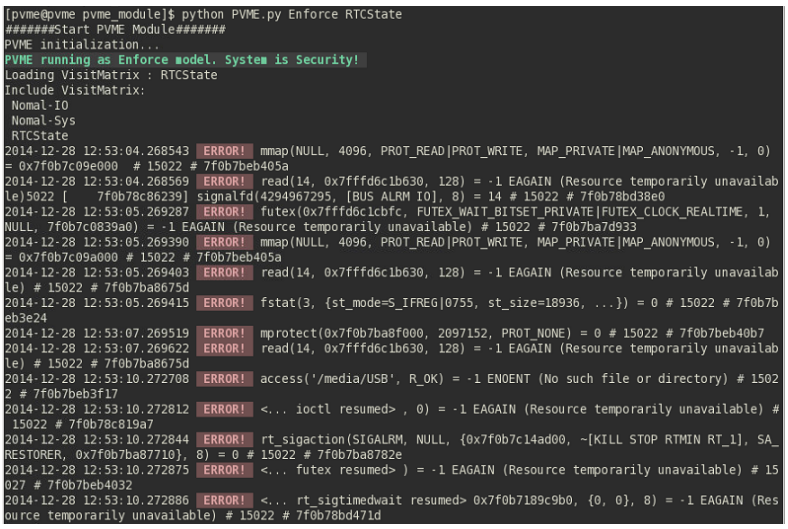

Figure 8. Simulation of preventing the RTSCtate attack using the PVME model.

The results suggest that the PVME model can efficiently prevent the network stack overflow operation caused by malicious code and protect the RTCState not to be modified during the information extraction stage of RTCState attacks. In summary, the PVME model can protect the system from being compromised by the remaining malicious pointer to ensure the RTC status not to be influenced after the illegal extraction of the U disk.

7. Conclusions

Through the PVME model simulation experiments, we have obtained the performance results as shown in Figure 9.

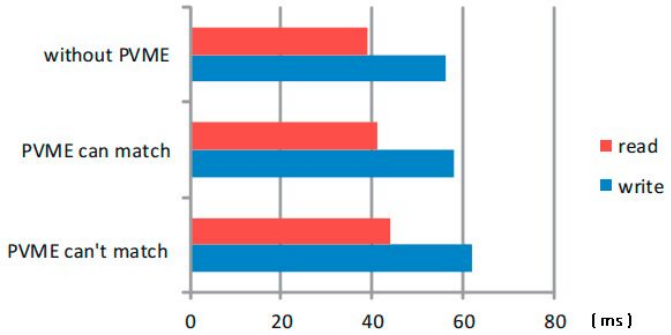

Figure 9. System costs.

As shown in Figure 9, the average time consumed by read and write operation in system call is 39.2 and 56.8 microseconds respectively without the PVME model. However the average time taken by a read and write operation requires an increase of 4% if the system satisfies the rules of the PVME model and if not an increase of 8%. Therefore the PVME model can efficiently reduce the possibilities of the VM escape attacks in the cloud environment without a significant impact on the system performance.

We add two new access properties (execute (e^*) and control (c^*)) to the PVME model and formulate several rules for different VM states in this paper. However, the specific application of the two properties used in the system call level is not provided in this paper. In the future, we plan to investigate the application of the two properties in the virtual machine and virtual machine system call level. While the VM level pays more attention to the Hypervisor control, the system call level needs to focus more on memory mapping, process protection, system reading and writing, and other aspects.

Author Contributions: Jiang Wu and Zhou Lei conceived and designed the PVME model; Jiang Wu performed the experiments; Jiang Wu wrote the paper; Shengbo Chen and Wenfeng Shen provided technical support and revised the paper.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Wei, L.; Zhu, H.; Cao, Z.; Jia, W.; Vasilakos, A.V. SecCloud: Bridging Secure Storage and Computation in Cloud. In Proceedings of the 2010 IEEE 30th International Conference on Distributed Computing Systems Workshops, Kozani, Greece, 21–25 June 2010; pp. 52–61.
2. Wei, L.; Zhu, H.; Cao, Z.; Dong, X.; Jia, W.; Chen, Y.; Vasilakors, A.V. Security and privacy for storage and computation in cloud computing. *J. Inform. Sci.* **2014**, *258*, 371–386. [CrossRef]
3. Ali, M.; Khan, S.U.; Vasilakos, A.V. Security in cloud computing: Opportunities and challenges. *J. Inform. Sci.* **2015**, *305*, 357–383. [CrossRef]
4. Fernandes, D.A.; Soares, L.F.; Gomes, J.; Freire, M.; Inacio, P.R. Security issues in cloud environments: A survey. *Int. J. Inform. Secur.* **2014**, *13*, 113–170. [CrossRef]

5. Modi, C.; Patel, D.; Borisaniya, B.; Patel, A.; Rajarajan, M. A survey on security issues and solutions at different layers of Cloud computing. *J. Supercomput.* **2013**, *63*, 561–592. [CrossRef]
6. Grobauer, B.; Walloschek, T.; Stocker, E. Understanding cloud computing vulnerabilities. *IEEE Secur. Priv.* **2011**, *9*, 50–57. [CrossRef]
7. Kazim, M.; Masood, R.; Shibli, M.A.; Abbasi, A.G. Security aspects of virtualization in cloud computing. In Proceedings of the 12th IFIP TC 8 International Conference on Computer Information Systems and Industrial Management, Krakow, Poland, 25–27 September 2013; pp. 229–240.
8. Borisaniya, B.; Patel, D. Evasion resistant intrusion detection framework at hypervisor layer in cloud. In Proceedings of the International Conference on Advances in Communication, Network, and Computing, Chennai, India, 21–22 February 2014; pp. 748–756.
9. Khan, A.A. *Isolation of Private Network from Internet, Secure Internet Virtual Environment*; Foundation of Computer Science: New York, NY, USA, 2015.
10. Yang, M.; Li, Y.; Jin, D.; Zeng, L.; Wu, X.; Vasilakors, A.V. Software-Defined and Virtualized Future Mobile and Wireless Networks: A Survey. *J. Mob. Netw. Appl.* **2015**, *20*, 4–18. [CrossRef]
11. Singh, A.; Chatterjee, K. Cloud security issues and challenges: A survey. *J. Netw. Comput. Appl.* **2017**, *79*, 88–115. [CrossRef]
12. Pearce, M.; Zeadally, S.; Hunt, R. Virtualization: Issues, security threats, and solutions. *J. ACM Comput. Surv. (CSUR)* **2013**, *45*, 17. [CrossRef]
13. Islam, T.; Manivannan, D.; Zeadally, S. A classification and characterization of security threats in cloud computing. *Int. J. Next-Gener. Comput.* **2016**, *7*, 1–17.
14. Liu, F.; Ge, Q.; Yarom, Y.; Mckeen, F.; Rozas, C.; Heiser, G.; Lee, R.B. CATalyst: Defeating last-level cache side channel attacks in cloud computing. In Proceedings of the 2016 IEEE International Symposium on High Performance Computer Architecture (HPCA), Barcelona, Spain, 12–16 March 2016.
15. Han, Y.; Chan, J.; Alpcan, T.; Leckie, C. Using Virtual Machine Allocation Policies to Defend against Co-Resident Attacks in Cloud Computing. *IEEE Trans. Dependable Secur. Comput.* **2017**, *14*, 95–108. [CrossRef]
16. Sze, W.K.; Srivastava, A.; Sekar, R. Hardening OpenStack Cloud Platforms against Compute Node Compromises. In Proceedings of the ACM on Asia Conference on Computer and Communications Security, Xi'an, China, 30 May–3 June 2016; pp. 341–352.
17. Liu, Q.; Wang, G.; Weng, C.; Luo, Y.; Li, M. A Mandatory Access Control Framework in Virtual Machine System with Respect to Multilevel Security II: Implementation. *China Commun.* **2011**, *7*, 137–143. [CrossRef]
18. Zhu, H.; Xue, Y.; Zhang, Y.; Chen, X.; Li, H. V-MLR: A Multilevel Security Model for Virtualization. In Proceedings of the International Conference on Intelligent Networking and Collaborative Systems, Xi'an, China, 9–11 September 2013; pp. 9–16.
19. Xue, H.; Zhang, Y.; Guo, Z.; Dai, Y. A Multilevel Security Model for Private Cloud. *Chin. J. Electron.* **2014**, *23*, 232–235.
20. Liu, J.; Li, Y.; Wang, H.; Jin, D.; Su, L. Leveraging software-defined networking for security policy enforcement. *J. Inform. Sci.* **2016**, *327*, 288–299. [CrossRef]
21. Yan, Z.; Li, X.; Wang, M.; Vasilakors, A.V. Flexible Data Access Control based on Trust and Reputation in Cloud Computing. *IEEE Trans. Cloud Comput.* **2015**, *PP*, 1. [CrossRef]
22. Bell, D.E.; La Padula, L.J. *Secure computer System: Unified Exposition and Multics Interpretation*; MITRE Corporation: Bedford, MA, USA, 1976.
23. Elhage, N. Virtunoid: Breaking Out of KVM. Available online: https://nelhage.com/talks/kvm-defcon-2011.pdf (accessed on 27 May 2017).