IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 1, 2018, accepted May 10, 2018, date of publication May 15, 2018, date of current version June 20, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2837039

Research of Security as a Service for VMs in IaaS Platform

XUEYUAN YIN[1], XINGSHU CHEN[2], LIN CHEN[1], GUOLIN SHAO[1], HUI LI[1], AND SHUSONG TAO[1]

[1]College of Computer Science, Sichuan University, Chengdu 610065, China
[2]Cybersecurity Research Institute, Sichuan University, Chengdu 610065, China

Corresponding author: Xingshu Chen (chenxsh@scu.edu.cn)

This work was supported in part by the National Key Technology R&D Program of China under Grant 2012BAH18B05 and in part by the National Natural Science Foundation of China under Grant 61272447.

ABSTRACT With the rapid promotion and application of cloud computing technology in various fields, cloud computing security has become the focus of attention. To satisfy the virtual machine (VM) security requirements of communication access control, network anomaly detection, memory monitoring, and file antivirus in Infrastructure as a Service (IaaS) platform, a comprehensive protection framework with the capacity of defense-in-depth for tenant VMs was proposed in this paper, which employed three different layers to satisfy above security requirements of tenant business from the outside to the inside of the VM. At the first layer, a tenant domain model was abstracted and realized based on software defined networking (SDN), which was used to re-obtain the capacity for communication access control for VM traffic and ensure security isolation of different tenant business networks. Besides, to detect the network abnormality of tenant VMs, a traffic structure stability model was constructed according to the deviation degree between current and historical normal network traffic structure profile. At the second layer, the capacities of network access control and anomaly detection, the same as the capacities used in the first layer, which were provided based on VM granularity. At the third layer, to monitor the VM memory information, a VM security monitoring method with agentless based on online analysis of VM memory was proposed by employing physical memory analysis mechanism. Moreover, a file antivirus method named HyperAV for VM based on virtualization was given, which was constructed of a frontend and a rear end. HyperAV optimized the process of virus scanning by monitoring the sector change information of a running VM with low performance costs. The experimental results demonstrated the effectiveness and low performance costs of the proposed protection framework and the corresponding security mechanisms, respectively.

INDEX TERMS Cloud computing, network security, access control, virtual machine monitors, antivirus.

I. INTRODUCTION

According to the CSA and NIST security guidance for cloud computing and virtualization technologies, Virtual Machines (VMs) in Infrastructure as a Service (IaaS) environment based on shared responsibility model involves a lot of security vulnerabilities [1]–[4]. Typically, the VMs may communicate with each other over an internal software backplane that lead to the traffic between them cannot be access controlled and monitored by standard network-based security controls. Moreover, the hidden processes are implanted in VMs maliciously may steal user confidential information. Furthermore, the VM operating system (OS) also faces the risk of viruses, which are hosted in the traditional OS maliciously. Proper information securities are suitable for cloud should be deployed in different locations to satisfy above security requirements of tenant business that loaded in VMs.

To address the above issues, different corresponding solutions are proposed by industry, open source group, and academia. Firstly, to re-obtain the management and control capacities of VM traffic in a virtualized environment, the VEPA and VN-TAG technologies are led by HP and Cisco respectively, and a distributed Layer2∼Layer4 stateful firewall service by use of a distributed kernel-enabled firewalling is provided by VMware NSX. And the security group and firewall as a service by employing Netfilter/Iptables are implemented in OpenStack [5], while the existing network framework fails to provide or access the capabilities of scalable advanced network security services

(e.g., network anomaly detection, deep packet inspection). Besides, to detect network abnormal behaviors, many research works have been done from the point of view of traffic analysis at present. The existing methods are mainly based on misuse detection, which depends on specific characteristics of the network attack, traffic detection rules, and a known attack feature library. The collected traffic data will be compared with the rule in the feature library and abnormal alarm will be outputted if it matches any rule [6]. As the method relies on corresponding rules those must be written in advance for attacks, it has to maintain a large feature library. Moreover, in the aspect of VM memory monitoring, there are two methods, the one named external monitoring mechanism is that the VM status information (e.g., VM Memory workload) is gathered through specific APIs, which are provided by virtual machine monitor (VMM) (e.g., Xen, KVM), while the information can be gathered is limited to the given APIs severely and the method is not easy to expand and customize, such as AWS CloudWatch and OpenStack Ceilometer products. The other method employed monitor probes placed in VM OS, which can retrieve more status information about a business process, while the probes should be protected due to their location and hence they were vulnerable to be attacked. Meanwhile, the privacy concerns could not be neglected. Besides, in a virtualized environment Xiang *et al.* [7] pointed out that the key data structure such as VM kernel could be analyzed by means of semantic reconstruction in the virtualization layer. And Li *et al.* [8] detailed the VM introspection (VMI) technologies based on the method of semantic reconstruction and kernel data structure parsing to retrieve VM memory information (e.g., process list, loaded kernel modules) externally, while the anti-attack ability and robustness of this method are weak due to the relationship chain between kernel data structures in a VM may be destroyed by malicious codes. Furthermore, to ensure the security of VM, an antivirus software is always installed in every VM. Consequently, high antivirus costs and duplications of virus database updating cannot be avoided. To mitigate the problems, McAfee integrated antivirus mechanism into the underlying virtualization infrastructure and the virus scanning work are transferred to a dedicated server [9]. Oberheide *et al.* [10] proposes an antivirus mechanism constructed with a frontend running in a VM to acquire executable files and a backend outside VM to perform virus analysis operations. CloudAV extended [10] by adopting multiple antivirus engines on backend to detection virus in parallel, and the frontend inside VMs was responsible for sending files with unknown Unique IDentifier (UID) values inside a white list to backends [11]. By providing unified anti-virus engines of above solutions, the redundant virus database in every VM is eliminated.

To satisfy the above concerned security requirements including the communication access control, network anomaly detection, memory monitoring and file antivirus for VM, a comprehensive protection mechanism with the capacity of defense-in-depth for tenant VMs was proposed. The detailed discussion of the contents is organized as shown in Fig. 1.

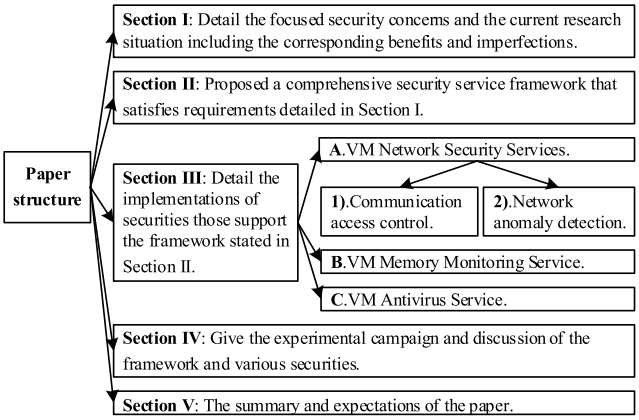


FIGURE 1. The paper structure.

Section I details focused security concerns in IaaS platform, which contains the communication access control, network anomaly detection, memory monitoring and file antivirus for VM. Then, different corresponding solutions are proposed by industry, open source group, and academia are discussed respectively, and corresponding benefits and imperfections are analyzed.

Section II proposes a comprehensive security service framework, which employs three different layers to orchestrate various security strategies to satisfy requirements as stated in section I.

Section III details the implementations of different securities those can support the framework discussed in section II, which include network security service, VM memory monitoring service and VM antivirus service respectively. And the part of network security service is consisted of mechanisms of VM communication access control and network anomaly detection.

Section IV gives the experimental campaign and discussion about the effectiveness and performance costs of the proposed framework and the various supporting securities.

Section V is the summary and gives a prospect to the research of this paper.

II. THE SECURITY SERVICES FRAMEWORK

To satisfy typically security requirements including network-related services to manage VM traffic and detect abnormal network behaviors, memory monitoring service and file antivirus service for VM in section I and orchestrate various corresponding securities, a security service framework was proposed according to the principles of security domain division and defense-in-depth strategy presented in Information Assurance Technical Framework (IATF), as shown in Fig. 2.

The framework employs three different layers to satisfy above security requirements. From the outside to the inside of the tenant VM, the layers are:

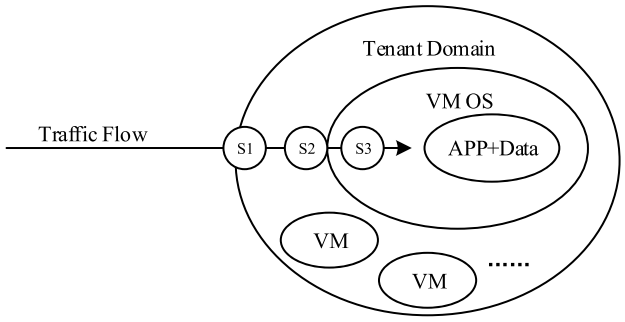


FIGURE 2. The defense-in-depth security service framework from the view of traffic flow of tenant VMs.

(1) S1 layer: to ensure tenant domain perimeter security. It provides capacities of managing tenant VMs network perimeters from the perspective of VM communication access control, and detecting network abnormal behaviors.

(2) S2 layer: to ensure VM perimeter security. It offers the capacities same as S1 layer based on VM granularity.

(3) S3 layer: to ensure VM OS internal security. It provides services of memory monitoring and file antivirus for VMs.

The related components deployed in IaaS platforms as shown in Fig. 3. The hardware layer provides physical CPU, memory, storage resources for virtualized IaaS environment. The hypervisor and security layer is put in charge of virtualizing above physical resources and providing enhanced security services respectively. And the security services are consisted of communication access control for tenant domain perimeter and VM perimeter, network anomaly detection, memory status monitoring, and files antivirus.

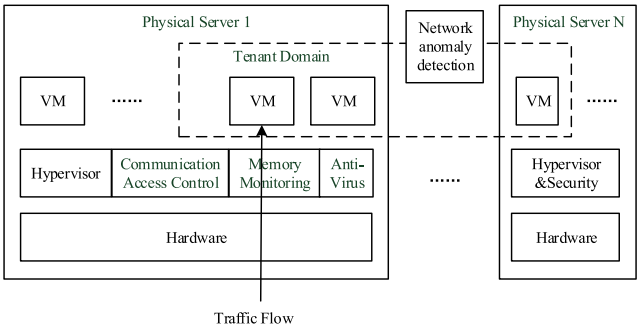


FIGURE 3. Related components of security service framework deployed in IaaS platforms.

III. THE SECURITY SERVICE IMPLEMENTATIONS

As the supports of the described framework in section II, this section details the implementations of different securities. Firstly, sub-section A discusses network security services those include communication access control and network anomaly detection respectively. Then, sub-section B states VM memory monitoring service. Finally, a VM antivirus service is described in sub-section C.

A. VM NETWORK SECURITY SERVICES

1) COMMUNICATION ACCESS CONTROL

To perform communication access control for VM and isolation of different tenants' business environment, a tenant domain model suitable for the IaaS platform was formally defined and designed. Then, an implementation framework of the model was given, which could segment the tenant network environment flexibly and manage fine-grained VM traffic, and eliminated configuration requirements for physical networks or physical firewalls effectively when network segmentation was performed.

Definition: Tenant Domain (*TD*). It contained 5 items. $TD = (TID, TNet, VMSs, ACRs, Cons)$.

(1) *TID*, represented a logical Layer 2 network tag. It satisfied the below constraint:

(*a*) $TID_i \neq TID_j, = if\ i \neq j$.

(2) *TNet*, represented a tenant virtual network environment that was constituted of different Layer 2 broadcast domains called *TSNet*. $TNet = \{TSNet_1, \ldots, TSNet_m\}$.

(3) *VMSs*, represented a VM set of a specific tenant. $VMSs = \{VMs_1, \ldots, VMs_m\}, VMs_i = \{vm_1, \ldots, vm_v\}$, and VMs_i represented all VMs within a subnet $TSNet_i$. If a VM vm_k that traffic was tagged *TID* and it was belonged into VMs_i within a subnet $TSNet_i$, then it was represented as $vm_k \in (TID \wedge TSNet_i \wedge VMs_i)$. The other entity *B* that did not belong to *VMSs* was represented as: $B \notin VMSs$.

(4) *ACRs*, represented a set of communication access control rules (ACRs) for a VM or VMs. $ACRs = \{ACR_1, \ldots, ACR_u\}$, $ACR_i = \{acr_1, \ldots, acr_z\}$, and ACR_i represented perimeter ACRs set of a subnet $TSNet_i$, and *acr* represented a specific ACR rule. According to *ACRs*, if $\exists acr\ (vmA, B)\ .\ action = pass$, then the communication behavior between *vmA* and *B* was permitted and denoted as: $vmA \overset{Y}{\longleftrightarrow} B$, else the communication behavior between them was denoted as: $vmA \overset{N}{\longleftrightarrow} B$.

(5) *Cons*: communication constraints for *VMSs*, which satisfied the following specific constraints:

(*b*) $vm_p \overset{Y}{\longleftrightarrow} B, if$

$$\begin{aligned}
&\exists acr_w \left(vm_p, B\right) \\
&\qquad .action = pass, \exists acr_z \left(vm_p, B\right). action = pass, \\
&vm_p \in (TID \wedge TSNet_i \wedge VMs_i)\ , \\
&\qquad B \in \left(TID \wedge TSNet_j \wedge VMs_j\right), \\
&acr_w \in ACR_i, acr_z \in ACR_j, w \neq z.
\end{aligned}$$

(*c*) $vm_p \overset{Y}{\longleftrightarrow} oB, if$

$$\begin{aligned}
&\exists acr_p \left(vm_p, B\right). action = pass, \\
&\qquad vm_p \in (TID \wedge TSNet_i \wedge VMs_i), \\
&B \notin VMSs, acr_p \in ACR_i.
\end{aligned}$$

(*d*) $vm_p \overset{N}{\longleftrightarrow} B, if$

$vm_p \in (TID \wedge TSNet_i \wedge VMs_i)$, *B* satisfied the conditions that excluded above (*b*) and (*c*).

According to the above constraints, the flowing security features could be obtained.

(1) By employing constraint (*a*), the isolation of tenant virtual network in IaaS environment could be ensured.

(2) By employing constraint (*b*), the traffic between different VMs within a same subnet or among different subnets belongs into a tenant domain could be managed.

(3) By employing constraint (*c*), the traffic between VMs in a tenant domain and other external entities could be managed.

(4) By employing constraint (*d*), the illegal communication between VMs in a tenant domain and other external entities could be blocked.

According to the formal description of *Tenant Domain*and our previous works [5], the implementation framework was constructed by employing network virtualization (NV), software defined networking (SDN) and network function virtualization (NFV) technologies, as shown in Fig. 4.

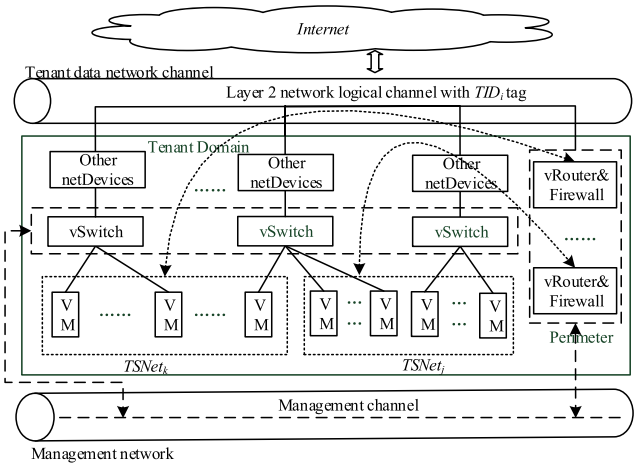


FIGURE 4. Tenant domain implementation model.

(1) Virtual communication channel, which was constructed by employing the VxLAN technology;

(2) Subnet configuration, which was relied on the virtual DHCP, router and switch appliances implemented by NFV technology such as the virtual appliance of OpenvSwitch.

(3) Tenant domain perimeter, which was consisted of a set of virtual routers and firewalls deployed on a virtual communication channel.

(4) Communication access control, depending on the ACRs which were configured in firewalls for constructing a tenant domain. Moreover, access control capacity (e.g. Layer 2∼ Layer 4 communication access control) of VM granularity was supported by employing SDN technology (e.g., Open vSwitch, Ryu controller and Openflow protocol), the related ACRs would be loaded in vSwitch components.

2) NETWORK ANOMALY DETECTION

To ensure the security of VM hosting a specific business (e.g., web service, mail service), a network abnormal detection mechanism is needed. First and foremost, the VM traffic needed to be captured and monitored by security appliances, in a virtualized IaaS platform, a specific VM traffic could be controlled and redirect into a specific security appliance to detect or filter by employing SDN and NFV technologies [5] (e.g., redirecting a packet from a specific VM to the IDS appliance through modifying the native destination MAC address in the packet to the IDS MAC address), the process was described as shown in Fig. 5.

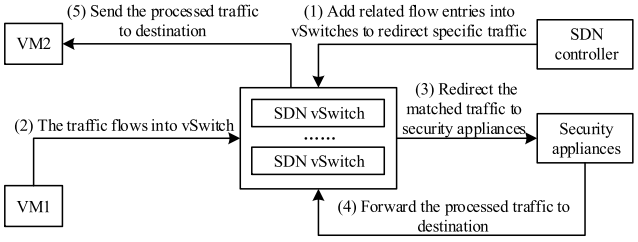


FIGURE 5. VM traffic redirection framework.

Furthermore, to detect VM network anomaly and eliminate the issues mentioned in section I, and to overcome the shortages of maintaining a large feature library and detect new network attacks difficultly, a concept of network traffic structure stability was proposed and described, which was based on the inherent stability of normal traffic attributes and the stability of a specific service. And a normal network behavior model for the server was profiled. Then, network abnormality could be detected by the deviation degree between current and historical normal network traffic structure profile.

a: FEATURES OF NETWORK TRAFFIC STRUCTURE STABILITY

According to the descriptions of entropy [12] (that describes the effective information of distribution changes of traffic characteristics) and correlation [13] (that detects traffic bursts such as DDoS), and after long-term observation and analysis of various attributes of network traffic, 8 features were selected to describe a traffic structure feature in a particular time window, as detailed below.

(1) SYN packet proportion (*syn*): a ratio of number of packets with SYN flag of the total number of packets.

(2) IP entropy (*ipentr*): entropy was calculated by a proportion of traffic, which corresponded to each IP address. In order to facilitate a calculation and comparison, an IP mapping was performed and it represented a logical mapping IP.

(3) IP correlation (*ipcorr*): a correlation between IP sequence with its traffic in a current time window and a previous time window.

(4) TTL distribution entropy (*ttl*): TTL entropy was calculated by a proportion of traffic corresponding each TTL.

(5) Port distribution entropy (*portsrv*): entropy of traffic corresponding to each port.

(6) Protocol distribution entropy (*porto*): entropy was calculated by a proportion of traffic, which corresponded to each protocol: TCP, UDP and ICMP.

(7) Packet length distribution entropy (*packet*): the entropy of traffic corresponding each packet length.

(8) Port access index (*portacc*): It depended on the number of server port, which was accessed by each client IP and corresponding probability of occurrence.

As an example, by analyzing network traffic of a mail server in a cloud data center, a statistical result of the above 8 features in 1000 consecutive time windows were shown in Fig.6.

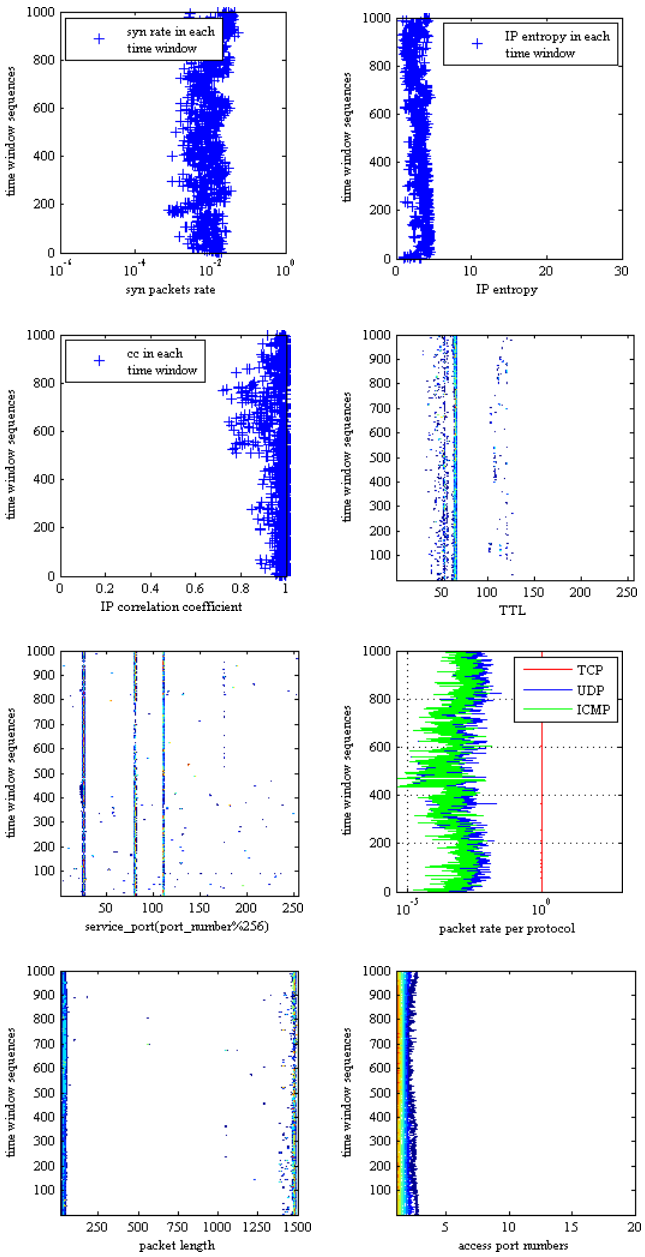


FIGURE 6. The 8 features of a network traffic structure.

Every feature has a relatively stable distribution on its value as below. The proportion of SYN packet is stable at 10-2 order of magnitude. The IP entropy is kept at around 4. The IP correlation is above 0.99. The TTL distribution is stable, the highest probability at 64 and 52. The service port numbers are distributed in 25 (SMTP), 80 (HTTP) and 110 (POP3), the feather matches the function of mail server. Protocols distribution including TCP, UDP and ICMP, are stable relatively. The packet length distribution shows that network traffic is mainly composed of long packets and short packets. The number of server ports accessed by vast majority of client IP is almost less than 3.

b: PROFILE A NORMAL NETWORK TRAFFIC MODEL

According to the normal network behavior profile changes periodically [14]–[16], a server network load and traffic are similar in adjacent time windows, and based on previous works [17], we profiled a normal network traffic model of current time that based on historical traffic of a former N time windows. In order to reduce the impact of the outliers in historical data on profiling normal model, the outliers in historical data based on the Grabs criterion [18] was eliminated.

We defined F was a set of network behavior 8 features: $F = \{syn, ipentr, ipcorr, ttl, portsrv, porto, packet, portacc\}$, N was a number of time windows before the current time, $f_b(x \in F)$ was a base feature value of network behavior within N time windows, ω was a stability coefficient of each feature in a network traffic structure, then defined P was a normal profile of network traffic structure, where a value of $f_b(x)$ was calculated by its definition of each feature.

$$P = \{\omega_x f_b(x) | f_b(x) = \frac{\sum_{i=1}^{N} x_i}{N+1}, x \in F\} \tag{1}$$

Stability coefficient ω_x reflected a stability of the feature x in normal situation. It indicated different influence of each feature on determining normal and abnormal, which was described by coefficient of variation c_v. The c_v was an absolute index that reflected a degree of data fluctuation, which did not depend on the size and dimension of multiple sets of data. It was suitable for determining a weight coefficient in multi-index comprehensive evaluation [18]. It was calculated as:

$$c_v = \frac{\sigma}{\mu} \tag{2}$$

where σ was a standard deviation, and μ was a mean. The smaller value of c_v, the smaller fluctuation of the data, the larger corresponding stability coefficient, thus there was an inverse relationship between c_v and ω_x. We used a logarithmic represented anti-correlation, and then ω_x was calculated as below:

$$\omega_x = -\ln\left(\frac{c_v(x)}{\sum_{i \in F} c_v(i)}\right) \tag{3}$$

The *diff(x)* was used to determine whether a network was abnormal or not. When *diff(x)* exceeded a certain threshold, it indicated that the current network traffic structure deviated from the normal profile to a greater degree, and thus judged it

in an abnormality. The *diff(x)* was calculated as follows, and $f_c(x)$ was a feature value of the current time window.

$$diff(x) = \sum_{x \in F} \omega_x |f_b(x)^2 - f_c(x)^2| \tag{4}$$

Based on the above discussion, compared with the existing misuse methods that based on rules, the method profiled for normal network behaviors without needing to maintain a large feature library and it could update the network behavior dynamically. Besides, according to the difference between normal network behaviors, this method could detect unknown attacks. Moreover, by employing the service access method based on SDN, and it had no effect to the existing business VM due to an agentless mode was adopted.

B. VM MEMORY MONITORING SERVICE

Physical memory records the real runtime information of OS and processes, thus, the running information of the processes can be revealed by analyzing the physical memory status, and particularly some malicious behaviors those only happening in memory can be captured. In a virtualization IaaS environment, a VM memory does not directly correspond to a physical memory, which is transparently managed and maintained by a virtual machine monitor (VMM) or a hypervisor. The VMM owns full access rights to VM and can access the physical memory of all VMs transparently. Thus, by deploying a mechanism of online VM memory analysis in the VMM, we can retrieve VM memory information without needing to deploy probes in the VM. The method could eliminate threats to the probes brought by malicious codes in VM.

According to existing research work of digital memory forensics [19]–[24], a method of cloud monitoring service based on real time online analysis of VM memory was proposed. To analyze a physical memory, a semantic analysis module for the VM OS was realized in the VMM layer, and a security monitoring service was composed of VMs that were created dynamically. Typically, the VM memory monitoring service framework based on KVM VMM was shown in Fig. 7.

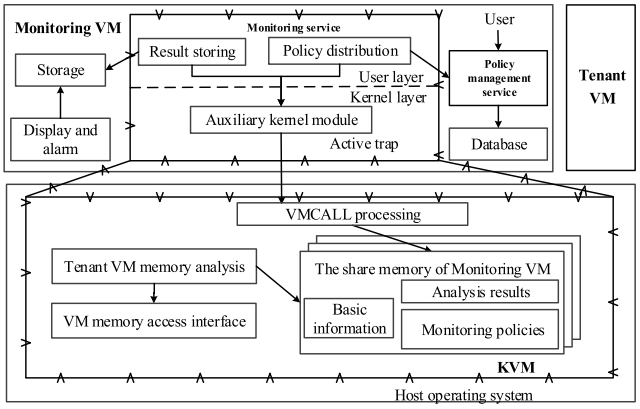


FIGURE 7. VM memory monitoring service framework.

The framework contains three components mainly: a user monitoring policy management service, a security monitoring service, and a VM memory online analyzing module.

(1) The user monitoring policy management service in a monitoring VM, which was responsible for managing (e.g., add, modify, delete, store policy) VM security monitoring policies (e.g., monitoring time intervals, specific monitoring indicators). Due to a tenant's VMs might be located in different servers and a server might contain different tenant's VMs, when a monitoring policy was configured for a VM, it was necessary to locate its position according to the database information firstly. Then, monitoring policies were sent to the monitoring VM in a same physical server.

(2) The monitoring service was consisted of two parts. The one is hosted in a monitoring VM, which was mainly consisted of three modules: policy distribution, result storing and auxiliary kernel module. The policy distribution module was used to receive the user's policies. The auxiliary kernel module was responsible for allocating shared memory blocks in the kernel of a monitoring VM for monitoring the tenant VMs in a same physical server. The shared memory structure was consisted of three parts: a basic information space for identifying a monitored tenant VM (e.g., UUID information), a policy space for a monitored tenant VM, and a space for storing analysis results. These memory blocks were efficiently shared among the application layer of the monitoring service, the auxiliary kernel module and the KVM. Due to KVM only allowing the codes running in VM kernel layer to trap into VMM by VMX instructions (e.g., VMCALL), thus the instructions would be executed after the necessary shared memory blocks had been allocated by the auxiliary kernel module, and then passed the addresses of those blocks to KVM. Finally, KVM would retrieve the information of VM identification and monitoring policies and store the analysis results.

The other is tenant VM memory analysis module, which was deployed in the KVM and in charge of analyzing a VM memory. When a VM was trapped into KVM because of exiting events (e.g., IO or interrupts), it was the right time to analyze the VM memory. Due to important kernel data structures of a VM (e.g., process control block, driver control block) was resided in memory, a real view of the VM current memory status could be reconstructed by parsing those kernel data structures. The steps of reconstruction semantic information of a VM from the KVM layer detailed as below.

First, we found the correlations of data structure. According to the correlations among the kernel data structures, the kernel data structures could be located. Took kernel data structure and their relationships in Windows 7 guest OS as an example to illustrate. By accessing the FS register, a KPCR structure value that stores a processor information can be retrieved. By analyzing the KdVersionBlock field in the KPCR structure and the LDR_DATA_TABLE_ENTRY structure, a kernel base address and an address of a driver double linked list and detailed driver module information can be retrieved. By analyzing the CurrentThread field in

the KPCR structure, information about the currently thread control block and its parental process control block can be resolved.

Secondly, we obtained the fingerprint features of data structure. Due to a relative chain of a kernel data structure might be destroyed by malicious codes, it brings negative effect to the reliability of memory analysis. To avoid the risk, by employing the fingerprint features of kernel data structure to locate them in memory directly in this paper, the algorithm 1 and algorithm 2 were used as below.

Algorithm 1 General Search Framework

input: *base,size. (The base represents base address, the size represents search range.)*
output: *Kernel object information*
Step 1. buffer←get_kernel_buf(buf_size,GFP_KERNEL); / buffer allocation*/*
*Step 2. sum_cmp←0; /*initialize an address offset for searching*/*
Step 3. while(sum_cmp is less than size) do
Step 4. if offset(size,sum_cmp) bigger than buf_size)
Step 5. cmp_size←buf_size
Step 6. else cmp_size←offset(size,sum_cmp)
Step 7.read_guest_memory(base+sum_cmp,buffer, cmp_size); / read a specified VM memory into the buffer*/*
Step 8.sum_cmp←sum_cmp + cmp_size
Step 9.for i←0 to cmp_size do
Step 10.if(IsRealSomeKernelObject(&buffer[i])) / According to fingerprint features, judge whether a kernel data of the current VM is the kernel data structure to be searched or not */*
Step 11. ShowSomeKernelObjectIn-fo(&buffer[i],vcpu) / Output related information of the located kernel data structure */*
Step 12. end if
Step 13. end for
Step 14. end while

The *IsRealSomeKernelObject* was an algorithm for identifying a specific kernel data structure. For a process control block (EPROCESS), referring to the existing work [20]–[22], we selected below fingerprint features to locate EPROCESS as shown in Table 1.

TABLE 1. The selected EPROCESS fingerprint features.

Field	Fingerprint
Pcb. Header.Type	0x03
Pcb.Header.Size	0x26
Pcb. DirectoryTableBase	%0x20 == 0
Pcb. ThreadListHead	> 0x80000000
CreateTime	>0
ExitTime	==0

Based on the above features, the implementation of the *IsRealSomeKernelObject* algorithm detailed as below. After finding the process control block in a tenant VM, the information about the process (e.g., modules list loaded, file list opened and port list listened) could be analyzed according to the data structure of the process control block.

Algorithm 2 The Process Control Block Identification

Input: *pAddr. (The buffer address that points a VM kernel space memory)*
Output: *Boolean. (Whether a data pointed by the current address is a process control block or not)*
Step 1. p←(PWIN7_EPROCESS_32)pAddr; / Convert virtual address to a type of process control block pointer */*
Step 2.if(p->Type! = 0x03||p->Size! = 0x26|| p->DirTableBase%0x20! = 0|| (p->ThreadListHead.Flink)<0x80000000|| (p->ThreadListHead.Blink)<0x80000000 || (p->ExitTime.LowPart! = 0 && p->ExitTime.HighPart! = 0))
*Step 3. return false; /*According to fingerprint features, the current data does not represent a process control block */*
Step 4. end if
Step 5. return true.

Based on the above discussion, by introducing the memory analysis mechanism and hardware virtualization technology, the method was able to obtain the runtime information of the guest virtual machine transparently. Besides, due to the monitoring agent was not needed in the VM and without any modifying to VM operating system, the security and portability of the method were enhanced.

C. VM ANTIVIRUS SERVICE

Traditional antivirus mechanisms require antivirus software to be installed in VMs, which may bring performance overhead during virus scanning and virus database updating. To solve the problem, we presented a framework called HyperAV, which was constructed with a lightweight frontend agent running inside VMs and a backend service to provide antivirus detection capability as shown in Fig.8. HyperAV was constructed with a guest agent called HyperAV Tools, QEMU front inside host server and HyperAV backend components deployed on separate servers. HyperAV provides following functions:

(1) Provide interactive interfaces for administrators/tenants to select files those will be scanned for antivirus;

(2) Fetch the physical sector locations of files and capture system events of file operations.

After files have been chosen for antivirus by administrators/tenants, HyperAV Tools fetch sector locations of above files and send sector numbers to HyperAV Backend. When HyperAV Backend receives the request from HyperAV Tools,

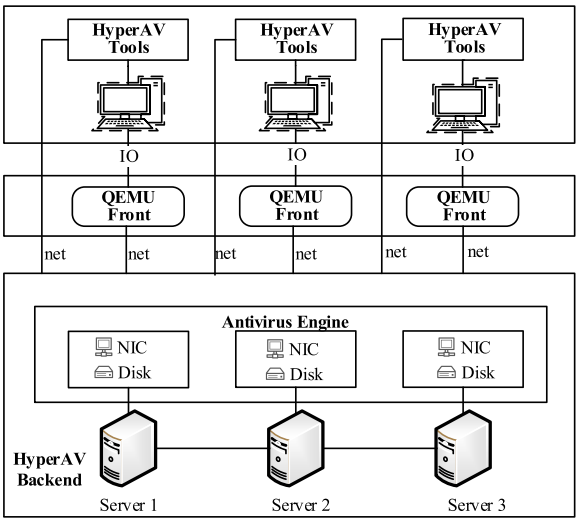


FIGURE 8. The framework of HyperAV.

HyperAV Backend adjudges whether the contents of these sectors have been changed from the last antivirus time firstly. If not, the last scanning result is directly returned to HyperAV Tools. Otherwise, HyperAV Backend issues a request to QEMU front, the QEMU front implemented a socket server and returns the contents of the sectors after receiving a request from HyperAV Backend. After the contents of these sectors are reconstructed as a file on HyperAV Backend, antivirus engines are called to scan the content of the files, then the scanning result is returned to HyperAV Tools. To monitor the sector change information of guest, QEMU emulator on host is modified so that sector numbers written by guest are sent to HyperAV Backend in real time, thus HyperAV Backend has knowledge of the completely modified sectors of guest VMs. Besides, when a file is detected to be harmful, HyperAV provides isolation mechanisms inside QEMU to prevent these sectors being read by guest and a notification message is sent to HyperAV Tools to inform the VM administrator.

As described above, the features of HyperAV were concluded as follows:

(1) A more lightweight guest agent: an agent process inside guest VM provided interactive interface to VM administrators, fetched sector information and captured file events. HyperAV Tools did not need to read the contents of files for antivirus and compute a HASH value to adjudge whether files have been changed during the passing of time.

(2) Combined antivirus with virtualization platform: an underlying virtualization platform such as VMM was modified to monitor the sector change information of guest VMs. When a file inside guest needed to be scanned, HyperAV could directly adjudge whether the file had been modified since the last scan, without having to calculate the HASH value of the file.

(3) Provided access control at sector level: HyperAV introduced access control flags inside QEMU to judge whether the sectors were forbidden being accessed by guest VMs to keep malicious files isolated from a guest OS.

(4) Reconstructed file content out of VM: HyperAV Backend could directly issue read operations of sectors with QEMU front, and these contents were reconstructed as a file by HyperAV outside the VM, which could reduce overhead of the guest VM.

(5) Multiple antivirus engines were supported: HyperAV Backend could adopt multiple antivirus engines to provide scanning service for a single file in parallel to improve accuracy.

Based on the above discussion, the method different from other cloud-related antivirus researches such as CloudAV [10], HyperAV neither required a specific VM to read contents of files to be scanned, nor did it need a VM to compute an UID value of a file and send the file to a backend for virus analysis. Instead, HyperAV directly fetched content of files from an underlying virtualization infrastructure and kept an account of vector change information of disks attached to a VM. By employing the mechanisms, HyperAV had a faster speed by directly skipping scanning unchanged sectors of files, which could save the time of reading contents and computing UID values of files effectively.

According to the method stated above and due to the realization of HyperAV was cloud platform-related, we implemented the method on KVM with qcow2 image format as an example to illuminate, and took windows 7 as guest VM with NTFS file system adopted. The implementation framework was shown in Fig. 9, and each part of HyperAV was discussed as follows.

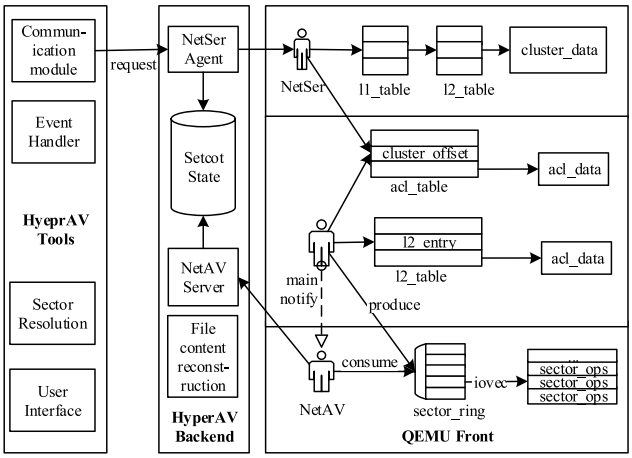


FIGURE 9. The implementation framework of HyperAV.

HyperAV Tools inside windows guest is constructed by four components:

(1) User Interface: provide command line interfaces to administrators/tenants to input related arguments, which include:

(a) Select files to be scanned for antivirus.

(b) Select files to be isolated by HyperAV.

(c) Select file paths to be watched, file modifications inside the path will be scanned in real time.

(2) Sector Resolution: after retrieving the input argument of the file to be antivirus or isolated, Sector Resolution component returns the physical sector locations of the file. For Windows NTFS file system, file data can be located either in Master File Table (MFT) or by calling DeviceIoControl with FSCTL_GET_RETRIEVAL_POINTERS and IOCTL_VOLUME_LOGICAL_TO_PHYSICAL as arguments individually.

(3) Event Handler: Monitor important file paths by utilizing windows event notification mechanism, modification inside these files will be scanned in real time.

(4) Communication Module: Provide network service to communicate with HyperAV Backend, passing the sector locations to be scanned or isolated.

QEMU Front is implemented on host, mainly provides interfaces to communicate with HyperAV Backend. The modification inside QEMU includes:

(1) Main thread: QEMU main thread handles guest events (e.g., network and IO emulation), and the following modifications were added:

(a) Fetches guest sector change information: Each time when guest issues a writing operation to disks, the operation can be concluded by a struct called sector_ops: {start_sector, sector_num, ops_size}, which is passed through QEMU to NetAV thread with a shared ring buffer. If host caches IO data inside memory, QEMU notifies NetAV to consume data inside a ring buffer at each flush of guest, otherwise, NetAV thread will be notified at each writing operation.

(b) Implement sector isolation: When QEMU accesses guest sector data, entries inside l1_table and l2_table are traversed. L2_entry which points out the location and attributes of the data cluster is modified to add two flag bits that represent whether the data clusters accessed by guest are forbidden being read or written. If the flag denotes forbidden, the acl_table will be accessed to get the permission.

(2) Acl_table: Point to clusters of access control data, as default, size of a qcow2 cluster is 64KB and guest sector size is 512B, two flag bits inside acl_data are needed to represent the forbidden access of reading and writing rights. Thus low 18 bits of guest sector number are used to traverse acl_data to get the access right, and the remainder of high bits (18-64) is used to traverse acl_table to get the location of acl_data. In addition, acl_table is accessed only when the forbidden bits inside l2_entry are set, so that normal access to sectors of guest will not be impacted.

(3) NetAV thread: Consumes data from the ring buffer produced by the main thread and sends these data to HyperAV Backend. The data inside ring buffer represent the sector numbers written by guest.

(4) NetSer: Accepts requests from HyperAV Backend and returns the sector data if needed by traversing QEMU l1_table and l2_table. If HyperAV needs to isolate sectors of guest, NetSer sets the corresponding flags inside l2_entry and acl_data.

HyperAV Backend mainly provides antivirus services for guest VMs and QEMU front and HyperAV Tools.

(1) NetAV Server: Accepts sector change information from NetAV thread of QEMU front and modifies states of these sectors as WRITTEN.

(2) NetSer Agent: Accepts antivirus request from HyperAV Tools and judges if the status of these sectors are marked as WRITTEN, which means the sector contents have been changed since last scanning. A request to NetSer inside QEMU front is sent to read the content of these sectors. Then contents of these sectors are reconstructed as a file for antivirus engines to scan. According to the scanning result, states of these sectors are modified as CLEAN or INFECTED.

IV. EXPERIMENTAL CAMPAIGN

According to Fig. 2 and Fig. 3, we employed five normal servers, several Gigabit Ethernet switches and Gigabit NICs to build IaaS platform. The key software parameters were: OpenStack Kilo as the IaaS management toolkit, KVM as the VMM and QEMU version was 2.3.0, OpenvSwitch as the virtual SDN switch and Ryu as the SDN controller, and CentOS 7.2 x64 release version as the host OS and its kernel version is 3.10.0-514.2.2.el7, the versions of guest VM OS are Windows 7 SP1 and Ubuntu 10.04. And key hardware parameters of physical server were: CPU was Intel(R)Xeon(R)CPU E5-2630 v3 @ 2.40GHz, memory capacity was DDR4 128GB. Another particular experimental parameters had been given in the corresponding section.

The deployment architecture of IaaS platform as shown in Fig. 10. By using independent network equipments to build different business function networks, including the management network, the storage network and the business data network, which are isolated from each other. For the security service network part, the security service is provided through the shared equipment such as a switch accessed into the service data network. The server in the DMZ is used to provide external users with the IaaS platform basic information display page and the tenant virtual resource management service interface.

A. NETWORK SECURITY SERIVCES

1) VM COMMUNICATION ACCESS CONTROL

According to the proposed theory model of tenant domain, the isolation network environment could be constructed for different tenants. The defense-in-depth business environment could be built effectively by performing the ACRs on firewalls (The method A in Fig. 11) or OpenvSwitch (The method B in Fig. 11) as shown in Fig.11.

For the method A in Fig.11, the functions of isolation and communication access control between different domains could be realized by employing different virtual routers configured different subnets and firewalls loaded various iptables/ebtables rules, which were similar to the settings of tradition physical routers and firewalls.

For the method B in Fig.11, it could be realized by employing OpenvSwitch, Ryu SDN controller and

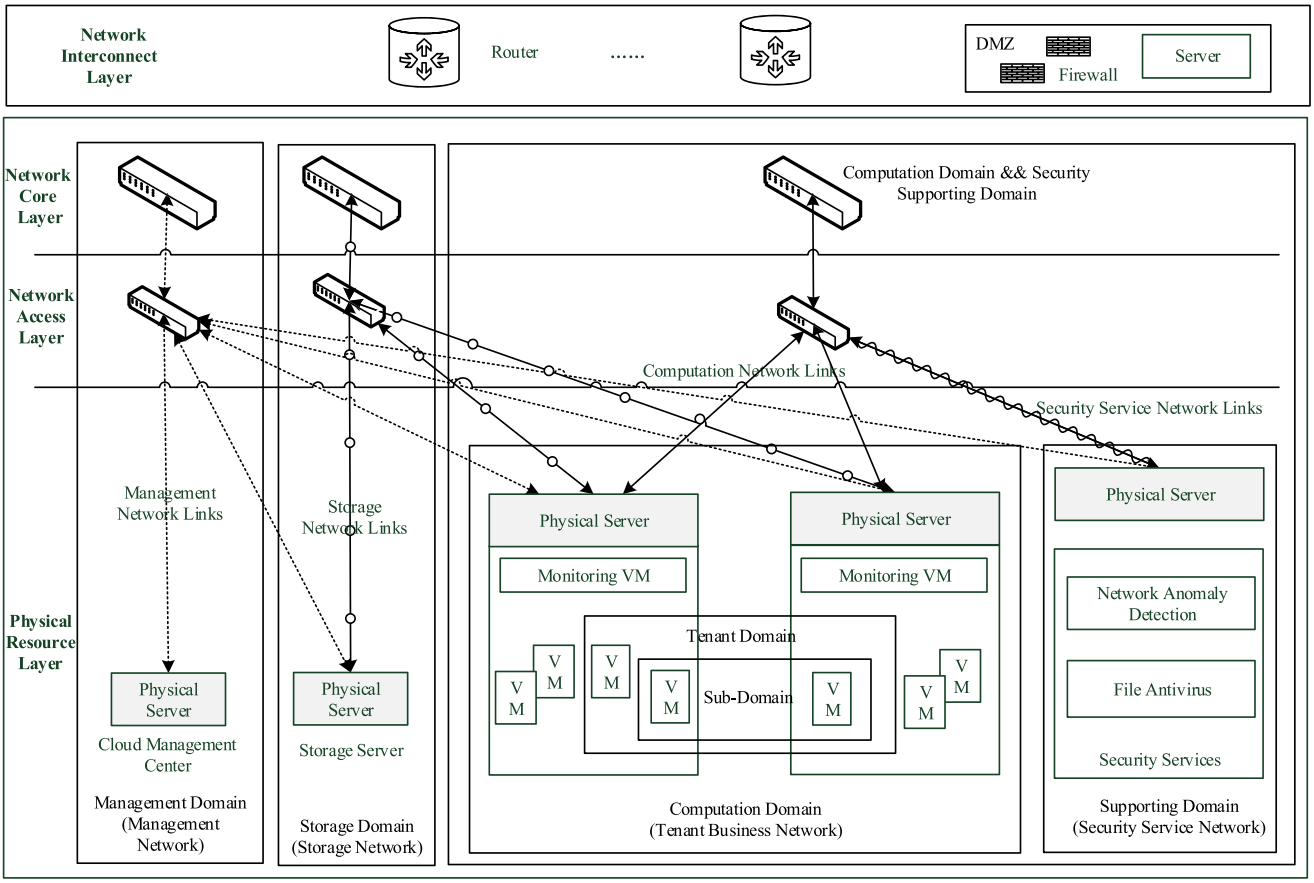


FIGURE 10. The deployment architecture of IaaS platform.

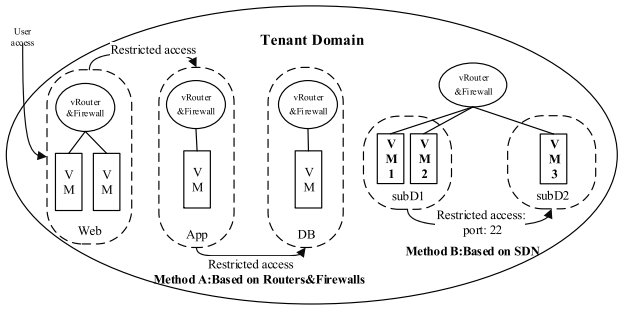


FIGURE 11. Defense-in-depth business environment for Tenants.

Openflow 1.3 protocol specification based on the principle of the Fig.4. The ACRs and relative experimental parameters as shown in Table 2.

TABLE 2. The key parameters of VMs.

VM	IP	MAC	Domain label
VM1	192.168.1.19	fa:16:3e:b0:ef:aa	subD1
VM2	192.168.1.22	fa:16:3e:9a:83:34	subD1
VM3	192.168.1.26	fa:16:3e:b2:d3:ec	subD2

The following abstract ACRs for subD1 sub-domain and subD2 sub-domain were defined as below.

(1) The rules for the subD1

'priority = 6, srcmac = {VM1,VM2}.MAC, dstmac = {VM1, VM2}.MAC, action = pass'

(2) The rules for the subD2

'priority = 5, srcmac = {VM1,VM2}.MAC, dstmac = VM3.MAC, dstport = 22, action = pass' 'priority = 5, srcmac = VM3.MAC, dstmac = {VM1,VM2}.MAC, srcport = 22, action = pass'

(3) The isolation rules for subD1 and subD2 *'priority = 4, srcmac = *, dstmac = *, action = drop'*.

Then, the following commands were executed on VM1 respectively.

(1) *'ping 192.168.1.22'*

(2) *'ping 192.168.1.26'*

(3) *'telnet 192.168.1.26 22'*

Both of the results of the 'PING' test for network interworking and isolation, and the 'Telnet' test between subD1 and subD2 were in line with expectations.

Furthermore, the performance cost of method B in Fig. 11 based on SDN discussed as below. We instantiated 30 VMs on a physical server and performed tests as shown in Table 3. The VMs named VA and VB within a same subnet on the server, and P1 was another physical server.

TABLE 3. Performance test cases of access control mechanism based on SDN.

Test case ID	Number of ACRs	Packet sending rate (Mbps)	Test time (S)	Test scenario
1	7	200	200	VA communicates with VB
2	7+650	200	200	VA communicates with VB
3	7	100	200	VA communicates with P1
4	7+100	100	200	VA communicates with P1

The load means and standard deviations of above four different test cases as shown in Table 4.

TABLE 4. Load means and standard deviations of four different test cases.

Test case ID	Load Mean		Standard Deviation	
	CPU(%)	Mem(%)	CPU	Mem
1	0.685	0.063	0.806	0
2	0.695	0.063	0.816	0
3	0.675	0.063	0.783	0
4	0.680	0.063	0.807	0

According to Table 3 and Table 4, the tenant network environment could be segmented flexibly only consuming little management cost and computing resource by employing the method B in Fig.11 based on SDN.

Besides, the both methods could eliminate configuration requirements for physical networks or physical firewalls effectively when network segmentation was performed, and have a prosperity of defense-in-depth according to different business isolation and security requirements, and the communication access control capabilities for VMs were provided at the same time.

2) NETWORK ANOMALY DETECTION

To verify the effectiveness of our method, we deployed our network behavior description system on a mail server in a data center to monitor its traffic flow. The change of traffic structures was observed by emulating attacks to the server. The system had the flowing features.

(1) The characteristics of traffic structures were statistics based on time series.

(2) A normal network behavior profile was constructed based on the historical data of the first 120 time windows. We eliminated the outliers in historical data based on the Grabs criterion, and a reference value was calculated based on average of the normal values, a stability coefficient was calculated based on the stability of the eigenvalues in the historical data, which was used to determine the weight of each feature value.

(3) The degree of deviation was calculated based on an actual traffic structure in a current time window and historical normal network behavior profile.

(4) The exception was alerted if a degree of deviation exceeded a certain threshold.

The value of a deviation threshold was different among different network environment, we calculated our threshold value by taking an average of the normal and exceptional values, which were calculated by the normal values and the outliers in the deviation historical data based on the Grabs criterion individually.

We implemented common network attacks to the mail server to test our system, and the result was shown in Table 5. For each type of attacks, three different attack scales were implemented. A total of nine experiments were conducted with each contained thirty attacks.

TABLE 5. The detection results of network attacks.

Attack type	Attack scale	Attack times	Detection times	Detection rate
Port Scan	120	30	5	16.7
Port Scan	240	30	30	100
Port Scan	600	30	30	100
SYN DoS	600	30	11	36.7
SYN DoS	1800	30	28	93.3
SYN DoS	3000	30	30	100
UDP Flood	600	30	24	80
UDP Flood	1200	30	28	93.3
UDP Flood	1800	30	30	100

The result showed that our system had an effective detection effect on port scanning, SYN DoS and UDP flood. The detection rate grows with the increasing attack scale and can finally reach 100% when attack scale reaches some critical value.

B. VM MEMORY MONITORING

We took Windows 7 as the VM OS to test the mechanism of VM memory monitoring.

1) OBTAIN THE PROCESS LIST

After we used a Tss Ht tool to hide a process named 'calc.exe' in the windows VM OS, the process could not be found through Microsoft command line interface by running 'tasklist' command inside the VM. Besides, the basic principles of a process hiding tool are: firstly, loading a driver which can perform DKOM operation in a VM kernel. The current activities of the process list from the current process control block can be retrieved. Then, after removal of 'calc.exe' process corresponding to the process control block from the active list of processes, so as to hide it when the process information is enumerated in the application layer.

While the process could be found under KVM by the proposed monitoring mechanism as shown in Fig. 12.

Due to the method did not rely on traversing process chain to get process information, but locating process control blocks directly from VM physical memory by using

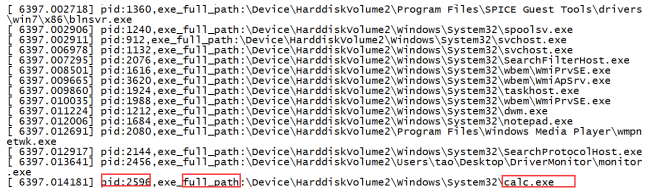


FIGURE 12. The retrieved process lists through KVM.

the unique characteristics of process blocks, so that hidden processes can be detected. Moreover, the method makes the by-pass means that take use of hooks to filter and forge OS data for deception of upper applications no longer valid, thus ensuring the acquisition of real information.

2) PERFORMANCE ANALYSIS

VM memory online monitoring service was a cyclical behavior, thus, by repeating a benchmark test procedure during the monitoring cycle, the average consumed time could be considered as the performance cost during a monitoring cycle. Table 6 gives the performance cost statistics of performing memory analysis, by using fingerprint features of EPROCESS within a monitoring cycle.

TABLE 6. The performance cost statistics.

Group	Consumed time(s)	Group	Consumed time (s)
1	0.378895	11	0.270886
2	0.278062	12	0.388277
3	0.337505	13	0.269555
4	0.282967	14	0.278654
5	0.274812	15	0.278007
6	0.278488	16	0.282598
7	0.278424	17	0.283713
8	0.279525	18	0.27218
9	0.271446	19	0.269447
10	0.269278	20	0.27852

The experimental results showed that the average performance cost caused by VM memory analysis was only 0.29s within a monitoring cycle. The performance cost would not affect the quality of tenant's services.

Furthermore, the method did not need to deploy any monitoring components within the VM, which had a higher security capacity than the methods employing monitor probes placed in VM OS. Besides, the security monitoring service runs in monitoring VMs, whose service capacities can be easily expanded by virtualization technologies.

C. VM FILE ANTIVIRUS

We implemented our prototype system on KVM with qcow2 image format, took windows 7 as guest VM with NTFS file system adopted, and took ClamAV as the antivirus engine. Fig. 13 shows the scanning results of a guest virus sample file, which was reconstructed by HyperAV Backend.

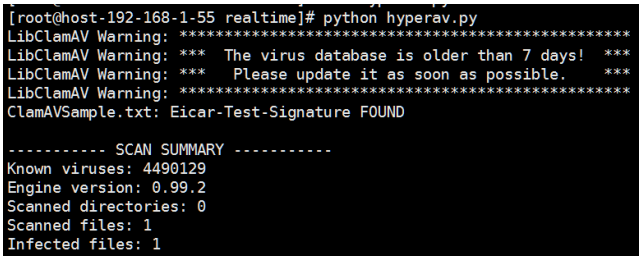


FIGURE 13. Scanning for virus by HyperAV backend.

Fig. 14 shows the effect of sector isolation for a file named 'ClamAVSample.txt'.

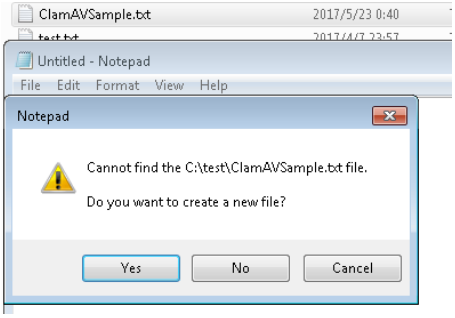


FIGURE 14. Sector isolation test for VM files.

When a file was performed sector isolation, due to the sectors of that file was marked as non-readable inside QEMU, zero values instead of the real data were returned to guest for a non-readable file.

Besides, Table 7 shows the performance overhead brought by HyperAV to monitor the sector change information of guest inside QEMU, which only affects the writing operations of guest. For each line inside the table, the performance loss value is calculated by $(normal - HyperAV)/normal$, and the result showed that the performance loss was about less than 1%.

TABLE 7. Comparison of performance test for IO write.

Test Arguments		Bandwidth (KB/S)		IOPS	
rw	bs	normal	HyperAV	normal	HyperAV
wr	4k	13960	13504	3490	3376
wr	32k	84732	81867	2647	2558
wr	128k	138121	137269	1079	1072
rdw	4k	1664	1644	416	411
rdw	32k	12417	12383	388	386
rdw	128k	39884	39594	311	309

Table 8 shows the time cost of HyperAV, MD5 computing and the time spent by ClamAV on scanning files. HyperAV Backend reconstructed the sector contents as a complete file to make ClamAV and MD5 computing work. For HyperAV, reading file contents was passed and the time was mainly

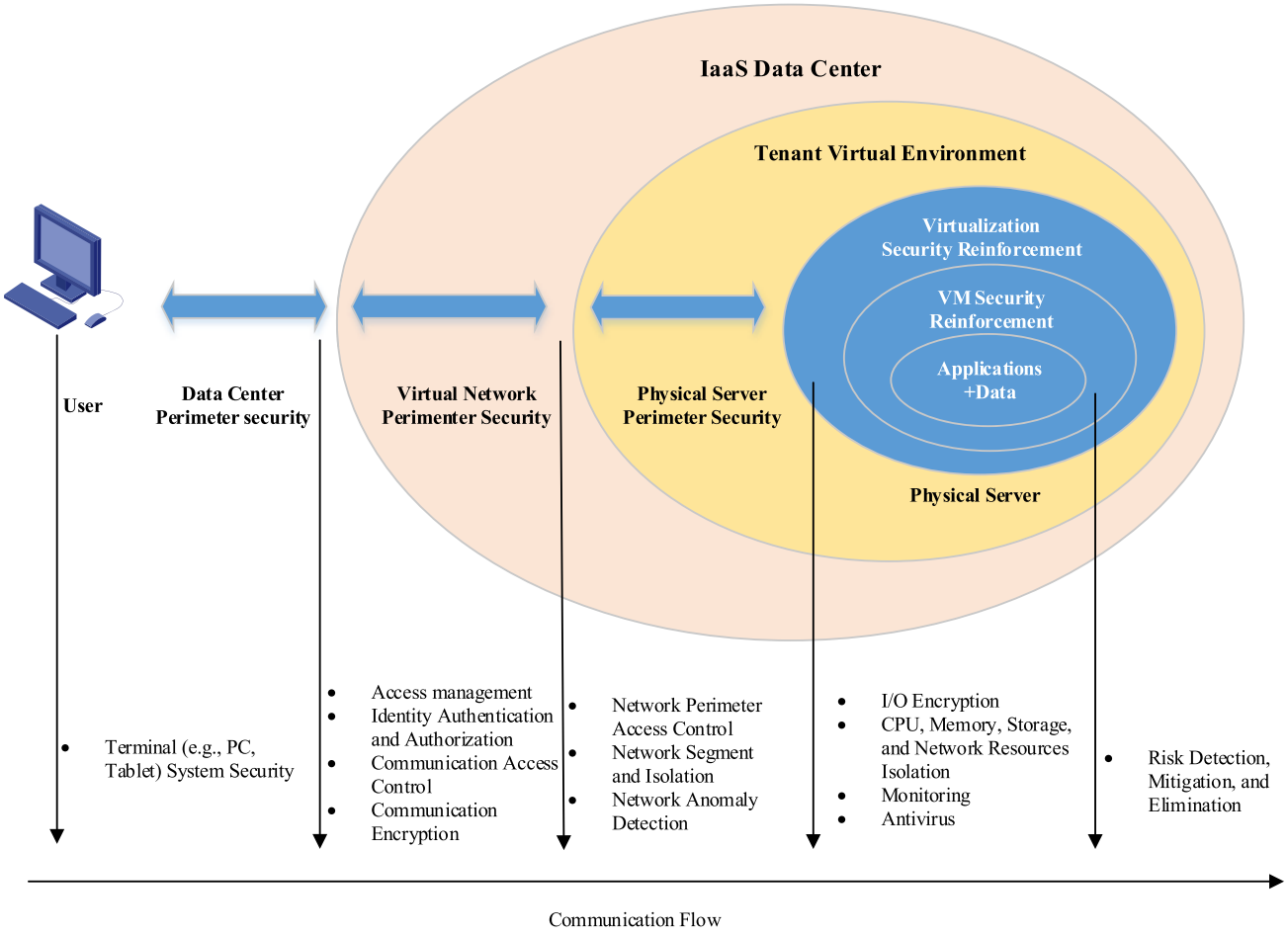


FIGURE 15. A more comprehensive security protection framework for a virtualized IaaS environment.

TABLE 8. Time cost comparison of general methods taken by antivirus software (time: 10us).

size\operation	ClamAV	MD5	HyperAV
100M	1762181	274085	18796
50M	926236	136295	10254
10M	895595	32509	2841
1M	843912	3143	313
500K	816426	1723	155
100K	786880	615	39

spent on querying the status of these sectors. It is obvious that MD5 computing can save much time compared with a real scanning process compared with ClamAV, and time spent by HyperAV is 10 times average less than computing a MD5 value.

V. CONCLUSION

In this paper, typical security requirements and corresponding solutions were given in IaaS environment were studied.

Firstly, using the principle of defense-in-depth strategy and security as a service, a layered safety protection framework for VMs was proposed to eliminate partial security risks, which contained communication access control, network anomaly detection, memory monitoring and files antivirus for VM. The framework employed three different layers and various security strategies to construct, which contained Tenant domain perimeter security, VM perimeter security and VM OS internal security. The capacities of managing VM communication and detecting network abnormal behaviors were provided in the layers of tenant domain perimeter and VM perimeter. In addition, the capacities of memory monitoring and file antivirus for VMs was provided in the layer of VM OS internal security. The framework was helpful to satisfy partial security requirements of tenant business dynamically.

Secondly, network security services were discussed, which contained services of communication access control and network anomaly detection. For the former, the essential factors of the tenant network domain were abstracted and a tenant domain model was constructed. The results showed that by employing the security domain model to construct network

architecture, the tenant network domain or sub-domain environment could be built with consuming little management cost and computing resource, which eliminated configuration requirements for physical networks or physical firewalls effectively when network segmentation was performed and have the prosperity of defense-in-depth according to the business isolation and security requirements, and the security isolation of different business networks could be ensured. For the latter, the anomaly detection method was proposed according to the degree of network traffic behavior deviating from the normal profile, the results showed that by employing the method, higher accuracy and lower false alarm were achieved than the method based on misuse detection, and the abnormal behaviors of SYN DoS and port scanning in the VM network could be detected quickly and effectively by means of traffic structure stability characteristics. Moreover, considering with new features of IaaS environment VM traffic, a dynamic access method of anomaly detection service based on SDN was realized. By employing the service access method, the network attack behavior could be detected to ensure the tenant VM security effectively, and without any negative effect on the existing business VM due to an agentless mode.

Thirdly, the problem of VM memory information monitoring in the cloud environment was studied, a VM monitoring method was proposed by online analyzing VM memory in the virtualization layer. The state information of the running virtual machine including active processes details and loading drivers details could be gained transparently. The results showed that the flexible and extensible capacity of VM state monitoring service could be provided at lower performance cost. Due to the monitoring agent was not needed in the VM and without any modifying to VM operating system, the security and portability of the method were enhanced. Furthermore, the monitoring VM could provide elastic monitoring capabilities by use of virtualization technologies. In addition, based on the monitoring data retrieved by the monitoring service, users could perform data analysis through other 3-party services, to judge whether the key business process code and data in a particular business VM are normal or not.

Finally, to solve the performance overload and resource consumption brought by antivirus software during virus scanning and virus database updating, a framework named HyperAV based on virtualization was proposed. HyperAV was able to provide antivirus capability for VM files with low performance overhead, and a mechanism of access control and isolation at the granularity of sector level was also presented. HyperAV optimized the process of virus scanning by monitoring the sector change information of a running VM on virtualization platform, which had a significant acceleration effect on the virus scanning process of VMs. The results showed that HyperAV was able to provide antivirus capability with low performance overload for VMs based on virtualization technologies, and did not need to deploy and update antivirus engines in every VM. Compared with traditional antivirus engines and other researches for cloud antivirus, HyperAV provided a more lightweight client agent, which reduced the process of reading file contents by computing UID values and thus the antivirus process was accelerated significantly. Meanwhile, the access control mechanism provided by HyperAV at the granularity of sector level was able to isolate virus files and protect user files effectively.

Based on the above research and practice of the framework and various security strategies, the framework was helpful to satisfy partial security requirements of tenant business in virtualized IaaS environment dynamically. Nevertheless, to ensure the security of IaaS platform based on shared responsibility model is a complex issue, besides the framework and various supporting securities discussed in this paper, other security measures should be further consideration and improvement such as identity authentication and authorization, communication encryption, partition and isolation of virtual storage resources, etc. A more comprehensive security protection framework for a virtualized IaaS environment as shown in Fig. 15 will be researched in future work.

REFERENCES

[1] G. Brunette and R. Mogull, "Security guidance for critical areas of focus in cloud computing v4.0," Cloud Secur. Alliance, Toronto, ON, Canada, Tech. Rep. Guidance v4.0, 2017, pp. 1–152.

[2] *SecaaS Implementation Guidance*, Cloud Secur. Alliance, Toronto, ON, Canada, 2012, pp. 1–333.

[3] R. Chandramouli, *Security Recommendations for Hypervisor Deployment on Servers*, NIST Special Publication, document 800-125A, Oct. 2015.

[4] R. Chandramouli, *Secure Virtual Network Configuration for Virtual Machine (VM) Protection*, NIST Special Publication, document 800-125B, Oct. 2015.

[5] L. Chen, X. S. Chen, J. F. Jiang, X. Y. Yin, and G. L. Shao, "Research and practice of dynamic network security architecture for IaaS platforms," *Tsinghua Sci. Technol.*, vol. 19, no. 5, pp. 496–507, Oct. 2014.

[6] B.-L. Cheng, H.-B. Zhou, and L.-H. Zhong, "Intrusion detection system based on anomaly and misuse," *Comput. Eng. Des.*, vol. 14, p. 019, Jul. 2007.

[7] G.-F. Xiang, H. Jin, and D.-Q. Zou, "Virtualization-based security monitoring," *J. Softw.*, vol. 23, no. 8, pp. 2173–2187, 2012.

[8] B. H. Li, K. F. Xu, P. Zhang, L. Guo, Y. Hu, and B. X. Fang, "Research and application progress of virtual machine introspection technology," *J. Softw.*, vol. 27, no. 6, pp. 1384–1401, Jun. 2016.

[9] McAfee, Santa Clara, CA, USA. (Apr. 2014.) *McAfee Management for Optimized Virtual Environments AntiVirus*. [Online]. Available: https://www.mcafee.com/cn/resources/data-sheets/ds-move-anti-virus.pdf

[10] J. Oberheide, E. Cooke, and F. Jahanian, "Rethinking antivirus: Executable analysis in the network cloud," in *Proc. 2nd USENIX Workshop Hot Topics Secur.*, 2007, Art. no. 5.

[11] J. Oberheide, E. Cooke, and F. Jahanian, "CloudAV: N-version antivirus in the network cloud," in *Proc. USENIX Secur. Symp.*, 2008, pp. 91–106.

[12] X. Du, Y.-J. Yang, and D.-X. Chang, "Network traffic supervision system based on feature distribution analysis," *Comput. Eng.*, vol. 6, p. 042, Jun. 2009.

[13] Z. M. Wang, "DDoS attack detection based on the stastical analysis," M.S. thesis, Dept. Comput., Yanshan Univ., Qinhuangdao, China, 2012.

[14] F.-L. Zhang *et al.*, "Prediction of network traffic based on traffic characteristic," *Comput. Sci.*, vol. 41, no. 4, pp. 86–89, 2014.

[15] Q. Y. Zhu, "Research on prediction model of network traffic," M.S. thesis, Dept. Comput., Xinjiang Univ., Urumchi, China, 2014.

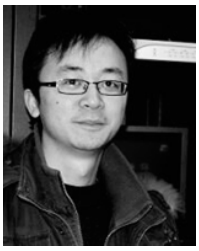
[16] X. M. Shao, "Research on campus network traffic measurement and performance optimization," M.S. thesis, Dept. Comput., HFUT, Hefei, China, 2014.

[17] G. L. Shao *et al.*, "Profiling structure-stability-based server traffic: Behavior models and system," *J. Univ. Electron. Sci. Technol. China*, vol. 1, p. 016, Jan. 2017.

[18] P. Q. Sun, "Correct selection of statistical criterion to eliminate outliers," *Meas. Techn.*, vol. 2013, no. 11, pp. 71–73, Nov. 2013.
[19] B. Dolan-Gavitt, A. Srivastava, P. Traynor, and J. Giffin, "Robust signatures for kernel data structures," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2009, pp. 566–577.
[20] I. Korkin and I. Nesterov, "Applying memory forensics to rootkit detection," in *Proc. ADFSL Conf. Digit. Forensics, Secur. Law*, 2014, pp. 115–141.
[21] A. Schuster, "Searching for processes and threads in microsoft windows memory dumps," *Digit. Invest.*, vol. 3, pp. 10–16, Sep. 2006.
[22] J. T. Sylve, V. Marziale, and G. G. Richard, "Pool tag quick scanning for windows memory analysis," *Digit. Invest.*, vol. 16, pp. S25–S32, Mar. 2016.
[23] J. Okolica and G. L. Peterson, "Extracting the windows clipboard from physical memory," *Digit. Invest.*, vol. 16, pp. S118–S124, Aug. 2011.
[24] Z. Lin, J. Rhee, X. Zhang, D. Xu, and X. Jiang, "SigGraph: Brute force scanning of kernel data structure instances using graph-based signatures," presented at the NDSS, 2011. [Online]. Available: https://www.researchgate.net/publication/221655382_SigGraph_Brute_Force_Scanning_of_Kernel_Data_Structure_Instances_Using_Graph-based_Signatures

XUEYUAN YIN was born in Yunnan, China, in 1988. He received the B.S. degree in computer science and technology from Sichuan University, Chengdu, China, in 2008, where he is currently pursuing the Ph.D. degree. His main research interests include cloud computing, network protocol analysis, and network security.

XINGSHU CHEN was born in 1968. She is currently a Ph.D. Professor and a Ph.D. Supervisor. Her research interests include cloud computing, cloud security, distributed file system, big data processing, network protocol analysis, and new media supervision. She is a member of the China Information Security Standardization Technical Committee.



LIN CHEN was born in Yibin, China, in 1983. He received the Ph.D. degree in computer science and technology from Sichuan University, Chengdu, Sichuan, China. His research interests include cloud computing, infrastructure as a service architecture, network security, and software defined networking.

GUOLIN SHAO was born in Jiangxi, China, in 1991. He is currently pursuing the Ph.D. degree with Sichuan University, Chengdu, China. His main research interests include network protocol analysis and network security.

HUI LI was born in Chongqing, China, in 1989. He received the M.S. degree in computer science and technology from Sichuan University, Chengdu, China. His research interests include cloud computing, virtualization technology, and security.

SHUSONG TAO was born in Xingyang, Henan, China, in 1989. He received the M.S. degree in computer science and technology from Sichuan University, Chengdu, China. His research interests include cloud computing, virtualization technology, and security.