Hindawi
Security and Communication Networks
Volume 2022, Article ID 1242576, 15 pages
https://doi.org/10.1155/2022/1242576

Research Article

VNGuarder: An Internal Threat Detection Approach for Virtual Network in Cloud Computing Environment

Li Lin,[1,2] **Huanzeng Yang**,[1] **Jing Zhan**,[1,2] **and Xuhui Lv**[1]

[1]*College of Computer Science, Faculty of Information Technology, Beijing University of Technology, Beijing 100124, China*
[2]*Beijing Key Laboratory of Trusted Computing, Beijing 100124, China*

Correspondence should be addressed to Li Lin; linli_2009@bjut.edu.cn and Jing Zhan; zhanjing@bjut.edu.cn

Received 25 October 2021; Revised 20 December 2021; Accepted 15 March 2022; Published 16 April 2022

Academic Editor: Chunhua Su

Edge-assisted Internet of things applications often need to use cloud virtual network services to transmit data. However, the internal threats such as illegal management and configuration to cloud platform intentionally or unintentionally will lead to virtual network security problems such as malicious changes of user network and hijacked data flow. It will eventually affect edge-assisted Internet of things applications. We propose a virtual network internal threat detection method called VNGuarder in a cloud computing environment, which can effectively monitor whether the virtual network configuration of legitimate users under the IaaS cloud platform has been maliciously changed or destroyed by insiders. First, based on the life cycle of cloud virtual network services, we summarized two types of internal attacks involving illegal use of virtualization management tools and illegal invocation of virtual network-related processes. Second, based on normal behavior of tenants, a hierarchical trusted call correlation scheme is proposed to provide a basis for discovering that insiders illegally call virtualized management tools and virtual network-related processes on the controller node of the cloud platform or the network node and compute node. Third, a trace-enable mechanism combining real-time monitoring and log analysis is introduced. By collecting and recording the complete call process of virtual network management and configuration in the cloud platform, and comparing it with the result of the hierarchical trusted call correlation, abnormal operations can be reported to the tenants in time. Comprehensive simulation experiments on the Openstack platform show that VNGuarder can effectively detect illegal management and configuration of virtual networks by insiders without significantly affecting the creation time of tenant networks and the utilization of CPU and memory.

1. Introduction

In the era of Internet of Things (IoT), more and more devices (e.g., sensors, terminals, household appliances, thermostats, televisions, automobiles, and production machinery) are connected to the Internet, and various edge-assisted IoT applications have emerged in industrial manufacture [1,2], medical healthcare [3,4], smart electric power grid [5], and privacy-aware federated learning system [6]. Due to the limited computing and storage capabilities of edge IoT devices, these applications often require virtual network services of cloud platforms to transmit data to be analysed or stored [7]. Network virtualization technology of cloud computing can achieve efficient reuse of network resources and centralized distribution of network traffic. In a cloud system, there are usually three kinds of networks [8] as follows. *External network* is used to realize the function of tenants' virtual machines accessing the Internet. *Management network* is used to provide internal communication between various components of cloud platform. *Data network* is a virtual network formed by data communication between virtual machines in cloud services, where *management network* is responsible for configuring and managing nodes in virtual network. Since management network and data network are completely controlled by cloud service providers, there are virtual network security problems such as malicious internal cloud managers using their own authority to directly change tenants' network configuration [9], intercept data transmitted in a virtual network [10], illegally read or write virtual network devices [9]. These internal threats will eventually affect the edge-assisted IoT applications.

Cloud security has always been a research hotspot in the field of network security [11]. The existing virtual network security protection researches mainly focus on the prevention of external threats of virtual network [12–14] and the solution of internal threats such as tampering of virtual machine image and storage faced by cloud compute nodes [15–17]. For the prevention of external threats to virtual network, the current work mainly focuses on data transmission security protection in virtual network. For example, Baik et al. [18] proposed a traffic monitoring method based on SDN architecture to find the illegal forwarding of tenant traffic and prevent the malicious tampering, reading and writing to virtual devices by attackers. Zhang et al. [19] proposed to use VPN technology to ensure data transmission security in virtual network and prevent malicious interception by external attackers. However, these researches ignore the virtual network internal threats caused by cloud platform managers. For the internal threats of cloud compute nodes, the current work mainly focuses on security protection such as virtual machine image and storage. For example, Miao and others [20] proposed to see all client-related operations made by KVM by adding a secure nested virtualization layer. In our early work [21], we proposed a virtualization protection framework supporting tracing to protect the important data of IaaS users stored on compute nodes from illegally accessing or maliciously damaging by malicious insiders. However, there have been different functions between network nodes and compute nodes in a cloud platform. Network node is responsible for controlling the virtual network of communication and virtual equipment configuration. The administrators and operators of cloud virtual network services with high authority can directly change a tenant's virtual network configuration, such as the tenant's three layer forwarding address, which may make the tenant's data forwarded to one attacker but the tenant do not know it.

There are three problems that should be solved in order to find the internal threats within virtual network in the cloud computing environment. Firstly, there are many virtual network devices such as *tap*s and *tun*s in the cloud platform. Malicious insiders may copy user data by overwriting the information of these virtual devices. Therefore, it is an important prerequisite for judging whether there is a threat by process monitoring of virtual network devices. Secondly, under the cloud virtual network service, tenants can configure and manage virtual networks using virtualization management tools. Because malicious insiders with higher privileges can call these virtualization management processes directly and then change the tenant's network via the hypervisor, these processes should be monitored. Thirdly, tenants often manage and configure their virtual networks by using visual API calls from cloud services. In this case, malicious insiders can also manage the network by calling these APIs to tamper with or impersonate tenants, so the API interfaces need to be monitored. In particular, it is not easy to distinguish whether the management of a virtual network by an insider comes from a legitimate tenant's request or from insiders' own malicious behavior.

To address the above issues, this paper proposes a virtual network internal threat detection approach in cloud computing environment called VNGuarder. In VNGuarder, based on in-depth analysis of network module source codes of cloud service, the API or function call relationships at different levels of virtual network service process of different nodes in cloud platform can be discovered. Exploiting finite state machine algorithm, a hierarchical trusted invocation association scheme based on tenants' normal behavior is proposed to provide a basis for discovering illegal invocation of insiders to virtualization management tools and virtual network-related processes on controller node or network/compute nodes of cloud platform. Meanwhile, a trace-enable mechanism to find the actual invocation behavior of virtual network from tenants, where some monitoring points are deployed in relevant processes and services of important nodes such as controller node, network node and compute node, and all invocation information collected by monitoring points in real-time are recorded in logs. Finally, a behavior matching algorithm is introduced to find malicious internal threat by comparing the actual trace with the results of hierarchical trusted invocation association.

Compared with the existing work, the main contributions of this paper are as follows.

(1) Compared with traditional data security transmission methods such as SDN-based traffic monitoring and VPN technology, this paper aims to solve virtual network security problems caused by malicious privileged insiders and has found two kinds of malicious management behaviors: illegal use of virtualization management tools and illegal use of virtual networks.

(2) Different from the existing single node monitoring schemes, this paper introduces a trace-enable mechanism that integrates real-time monitoring and log analysis, and comparing it with the hierarchical trusted invocation in tenants' normal behavior, which can improve the detection effect of insider threats.

(3) The authors have successfully implemented VNGuarder in Openstack platform and conducted several comprehensive experiments to evaluate the effectiveness and performance of VNGuarder. Experimental results show that VNGuarder can detect not only internal threats such as traditional single-process illegal calls, but also important internal threats such as modification of virtual network device configuration through virtualization tools, with only a small performance degradation.

The rest of this paper is organized as follows. Section 2 presents related work. Section 3 outlines problem statement, including threat model considered in this paper. Section 4 introduces VNGuarder in detail. Section 5 presents our experimental evaluation results. In Section 6 we provide a discussion on the comparison VNGuarder with our earlier work and other virtual network detection methods. In

Section 7, we conclude the paper and present some possible future work.

2. Related Work

Virtual network security under cloud computing environment has always been a research hotspot in the field of cloud security. Researchers have proposed some detection and protection schemes, including traffic monitoring based on SDN, monitoring at kernel. For example, Hussein et al. [22] proposed to use SDN technology to automate the collection of network traffic, network behavior, security events, and other information, so as to find out the traffic hijacking behavior of malicious insiders. Shao et al. [23] proposed a security model based on SDN technology. By using the SDN control layer, the virtual network and virtual network equipment were divided into independent security management domains, and corresponding security event information was collected and monitored. SDN is a kind of network virtualization technology. The separation of forwarding and control layer is helpful to find threats such as malicious change of virtual equipment and traffic theft under the virtual network, but the controller itself will still face internal threats of illegal management intentionally or unintentionally within the control scope. Carvalho et al. [24] proposed to use Nagios cloud computing platform monitoring module and complete monitoring of monitoring points in the form of plug-ins. Based on monitor of kernel, although it can effectively judge whether the related process modules are abnormal, real-time monitoring will consume a large amount of CPU resources, and such a single abnormal behavior is more likely to cause misjudgment and affect the experience of cloud services.

Insider threats are one of the most serious security challenges facing cloud computing [25]. Internal threats mainly come from cloud service providers, including the intentional or unintentional management and configuration of tenant data and tenant network by cloud service insiders. In terms of internal threat detection and prevention of tenant data, Johannes and others. [26] found that malicious cloud service insiders could steal user data by reading memory and file system information. Some researchers have come up with solutions to these security threats, for example, based on trusted computing, Yu et al. [27] combined trusted computing and cloud security by establishing TPM (short for Trusted Platform Module). Implementing a complete set of trusted systems for detecting and verifying VM identity information, Hussein et al. [28] proposed a framework to review and monitor the hard disk, CPU, and user data in the virtual machine image and protect the security of the virtual machine image in the cloud environment through an expert review method. Kansal et al. [29] proposed an early detection and isolation method called EDIP to mitigate insider attack behaviors. EDIP detects all legitimate clients in the system at the agent level and isolates innocent clients by migrating them to the attack agent. Singh et al. [30] proposed a general behavior-based insider threat detection method, where the behavior is characterized by user activity (such as logon-logoff, device connect-disconnect, file-access, http-url-requests, e-mail activity). But the above research focuses on solving the internal threats to data storage in the cloud environment, ignoring that malicious insiders can intercept tenant data by monitoring and modifying the tenant's virtual network.

To sum up, the existing virtual network security protection in cloud computing environment mainly focuses on external attackers and ignores insider threats. Therefore, this paper proposes an insider threat detection approach to deal with the security risk in management and configuration of virtual network rather than data transmission.

3. Problem Description

This section introduces application scenarios and threat model considered in this paper.

3.1. Application Scenarios. Figure 1 depicts a common scenario for IaaS cloud service. In IaaS deployment, a cloud user, which has been authenticated and authorized by CSPs, can manage or configure his virtual network by logging in his management interface on cloud platform (such as Openstack). When a user or compute service components' (such as nova) request is sent to virtual network control components (such as neutron-server) in controller node, the virtual network control component will invoke virtualization service process (e.g., agent process) by calling virtualization management tools and the corresponding remote procedure call API. Then a virtual network can be built up by all kinds of virtual network devices (e.g., Bridges, Virtual Switches) and hypervisor, where tenant can manage and configure the virtual network.

The normal startup process of cloud virtual network service by users is shown in Figure 2. After obtaining authorization, a user can manage his private network through cloud platform management interface, such as to create, delete, change, or configure the network services (e.g Firewall, Load Balancing, NAT) or network's core resources (such as Network, Subnet and Port). The user service interface will be invoked when a user manages his virtual network in the console. When the controller node receives the user requests, network service interface such as the Core of neutron-server API or the Extension API in controller node will determine whether to need to invoke a remote call interface in line with the request type. If necessary, the request is forwarded to the compute node of the cloud service or the management implementation interface of the network node through the remote call interface. According to different request services, the management implementation interface invokes the corresponding virtualization process (such as agent process) and finally completes the configuration of virtual network through the virtualization process.

3.2. Attack Model. Since users' private network is exposed to cloud service nodes in current cloud environment, insiders can not only pose security threats to virtual network through direct access but also threaten users' network security by directly calling cloud service-related API. On the

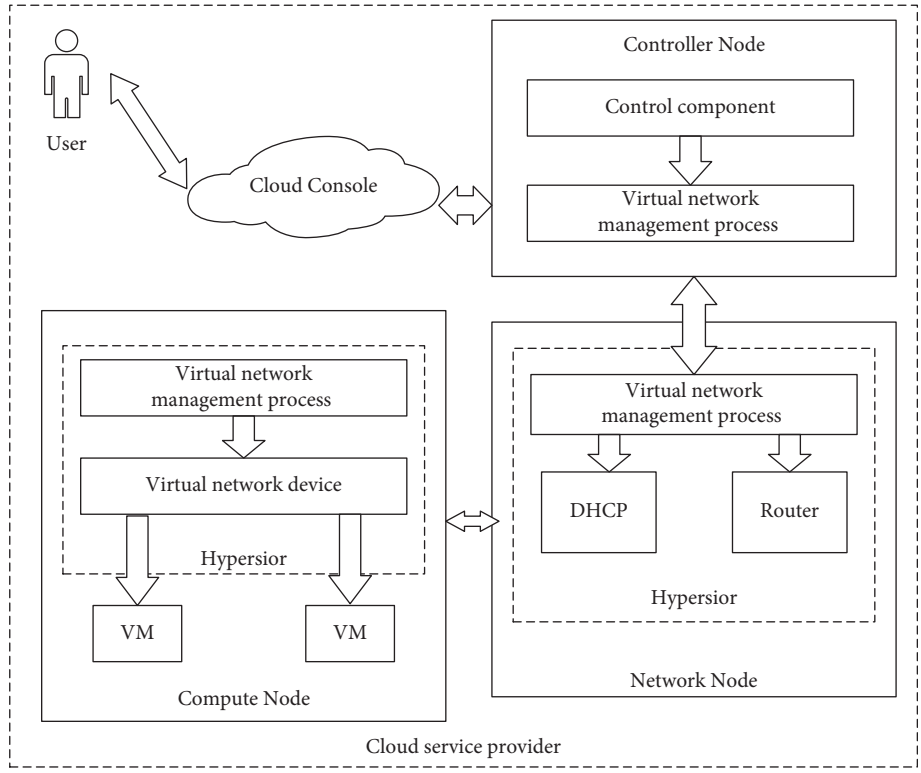

Figure 1: Network service in IaaS deployment.

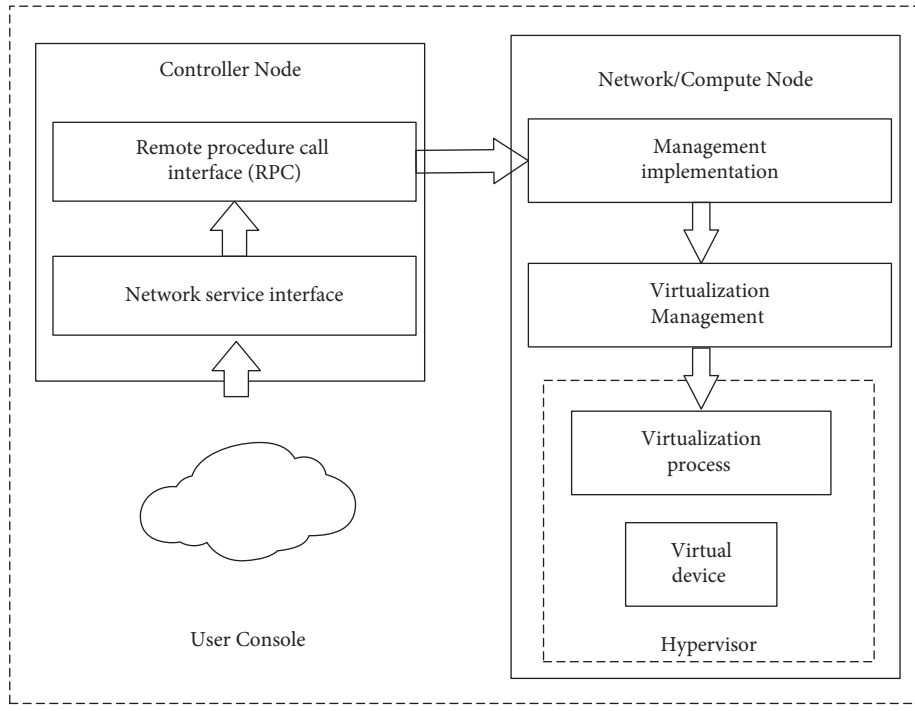

Figure 2: Normal startup process in IaaS network service.

mainstream platform, such as Openstack, a malicious insider can invoke different APIs at different nodes. For instance, a malicious insider can invoke the virtual network service management interface named neutral-server at controller node to manipulate the user's private network, or call libvirt virtualization management tool with virsh module to change the current user's virtual network, which cause legitimate users to not be able to access virtual network cloud service. For another example, a malicious insider can call the virtualization process qemu-kvm (short for quick emulator and

kernel-based virtual machine) in a compute node or a network node to enter the virtual device, illegally rewrite the virtual device, and forward the copy traffic to the designated virtual host in the virtual network, causing user data security issues. If you only monitor a single API interface or process level, there is no way to tell whether these actions are normal user behavior or are illegal malicious changes from malicious insiders.

There are two main concerns from the perspective of cloud tenants. One is whether his private network will be infiltrated and destroyed by malicious insiders. The other is whether the communication in the cloud network is safe. Since a users' management and configuration to his virtual network is directly exposed to cloud service provider, there must be security threats from insiders or untrusted cloud service providers. As shown in Figure 3 there are two kinds of insider threats in virtual network.

3.2.1. Insider Manages and Configures a User's Network by Making Unauthorized Calls to Virtualization Management Tools or Processes Related to the Virtual Network. Cloud platform administrators can use their administrator privileges to achieve the purpose of changing user network configuration and state by making unauthorized calls to virtualization management tools and related virtual network processes. For example, for qemu virtualization process, the command "*brctl add br br0, brctl add if br0 eth0* "can be used to create a bridge and add eth0 (represents the first Ethernet card) to the bridge. For virtualization management tool libvirt, Internal operators can use commands "*virsh net-udefine VBR", "virsh net-destroy VBR", "virsh net-edit VBR"* to delete, stop, and modify the specified virtual network. Apparently, unauthorized calls by these virtualization management tools or virtualization processes associated with the virtual network will pose a security threat to the virtual network.

3.2.2. Insider Attack a Users' Virtual Network by Changing Its Virtual Devices Configuration. The malicious cloud manager can directly configure the user's virtual devices (such as OVS) inside the cloud platform, including changes to the OVS port and network. When a user (e.g. edge devices in IoT) normally uses the cloud platform to store or transfer data, a malicious insider indirectly changes the storage address of the user's data or directly obtains the data by changing the configuration of the user's virtual device.

To detect the above two threats, a method called VNGuarder is proposed in this paper.

4. Design of VNGuarder

In this section, we will give the design of the proposed VNGuarder method, including its working principles and detailed functions.

4.1. Working Principles. For the above application scenario, the basic idea of VNGuarder is to build a hierarchical trusted invocation association by analysing the network module source code of cloud service. Based on the hierarchical trusted invocation association analysis, various user's trusted behaviors are constructed. Conduct and deploy information acquisition and monitoring modules at various node levels such as network service interface, remote call interface, management implementation interface, virtualization management interface, and virtualization process shown in Figure 2. Meanwhile, the actual management implementation process of cloud virtual network service is monitored in real time. All the calls of API and process of nodes at different levels are recorded. By matching the actual cloud service behavior acquired from acquisition and monitoring modules with trusted behavior of normal user, the illegal management and configuration of users' virtual networks by malicious insiders can be found. The overall workflow is shown in Figure 4.

4.2. Construction of Trusted Behavior

4.2.1. Description of Tenant Trusted Behavior Based on Finite State Machine. In VNGuarder, the user's interaction with his virtual network cloud service is analysed based on all kinds of interfaces, such as virtual network service interface, remote call interface, management implement interface, virtualization management interface, and so on. Mining multilayer APIs association in source codes, the trusted invocation behavior of each legitimate user using the virtual network service under cloud platform is obtained. The keywords of each node are created by the proposed hierarchical trusted invocation association method.

Different types of logical nodes are established according to the invocation relationship between different nodes. From the user entry node to the last implementation node, the behavior of each layer corresponds to the hierarchical trusted invocation association. For example, the management call process of OpenStack Neutron service to virtual network is shown in Figure 5. A user or compute node forwards a request information to controller node of the subserver (such as Neutron-server) by calling RESTful API or Nova-API, where whether to call core API or extended API depends entirely on the type of virtual network service. And then the request information is forwarded to the plugin, where whether to make RPC (short for Remote Procedure Call) or not also depends on the specific needs of the user. If any RPC is called, the request is forwarded to one compute node or the management implementation interface of network node. The management implementation interface calls the corresponding virtualization process (such as Agent) according to the user's management request.

There is an initial state in a normal hierarchical trusted invocation association process. When an input or jump condition is detected, it will jump to the next state. Therefore, the hierarchical trusted invocation association process of a legitimate user's can be described by a finite state machine. The FSM is described as a five-tuple as follows:

$$M = (Q, \sum, \delta, q0, F), \tag{1}$$

where $Q = \{q0, q1, \ldots, qn\}$ is a set of finite states. At any given moment, the network is in a certain state q_i; $\sum =$

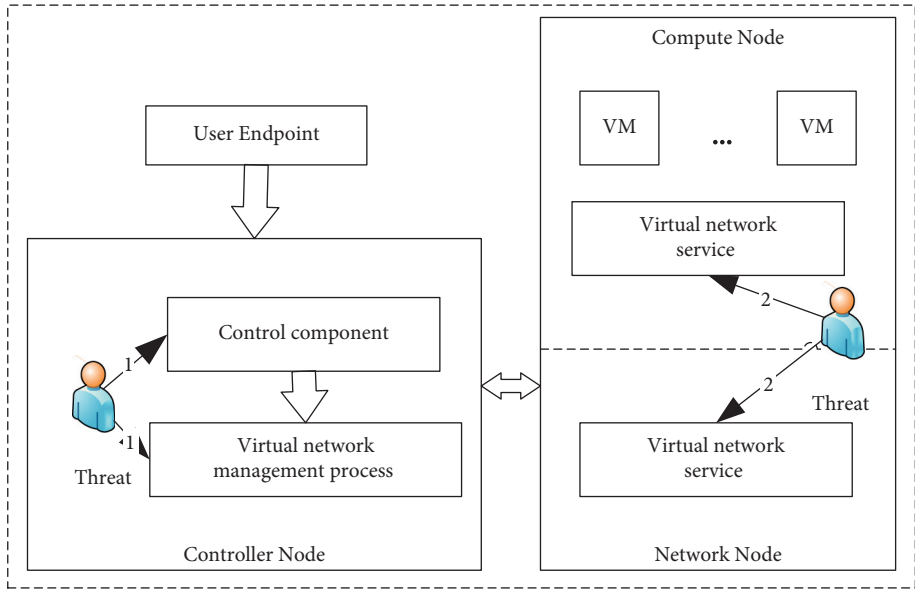

Figure 3: Potential insider threats in IaaS service deployment.

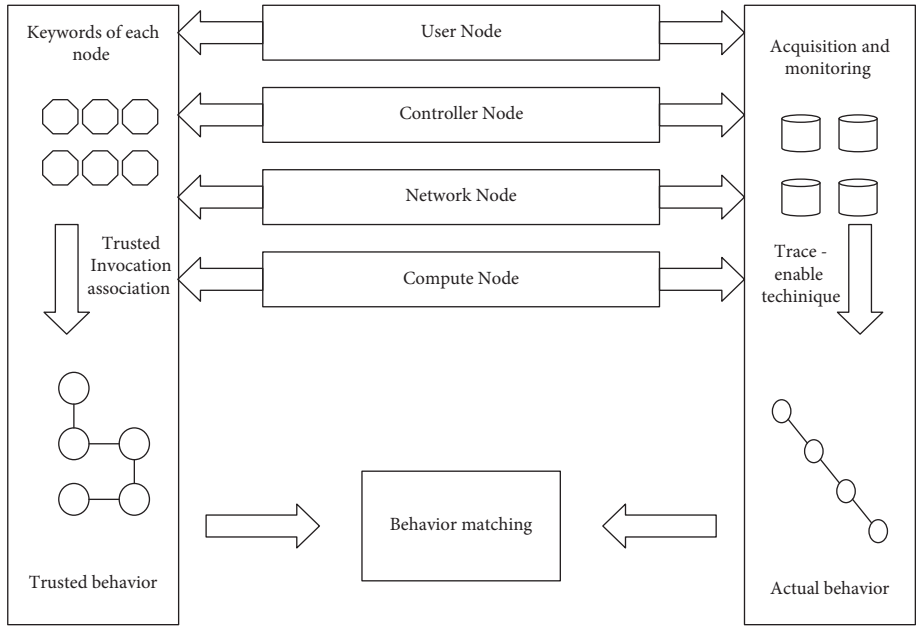

Figure 4: Workflow of VNGuarder.

$\{\sigma 1, \sigma 2, \ldots, \sigma n\}$ is a set of finite input σ_j. At any given moment, a given input j is received when the network behavior state changes. σ: $Q \times \sum \longrightarrow Q$ is state transfer function. That says, in a certain state, state after a given input into a new state decided by the state transition function. $q0 \in Q$ is the initial state of virtual network management behavior, from which the finite state machine begins to receive input. $F \in Q$ is the final state set of virtual network management behavior. A finite state machine does not receive new input after it has reached the final state.

The above FSM is specifically used to describe the trusted behavior of a normal user in virtual network cloud service. Taking the tenant's operation *useraction* on the virtual network as an example, the tenant's trusted behavior description based on finite state machine is shown in Figure 6.

4.2.2. Tenant Trusted Behavior Generation Based on Hierarchical Invocation Association Analysis. As stated above, a trusted behavior of a normal user in virtual network service is described as an FSM, where each state represents the users' legitimate invocation operations at each level in cloud service. To generate this FSM, the following steps need to be completed. First, for each level of cloud service, the relevant source code-based keywords must be extracted as a keyword database. Second, when a user's trusted invocation operation

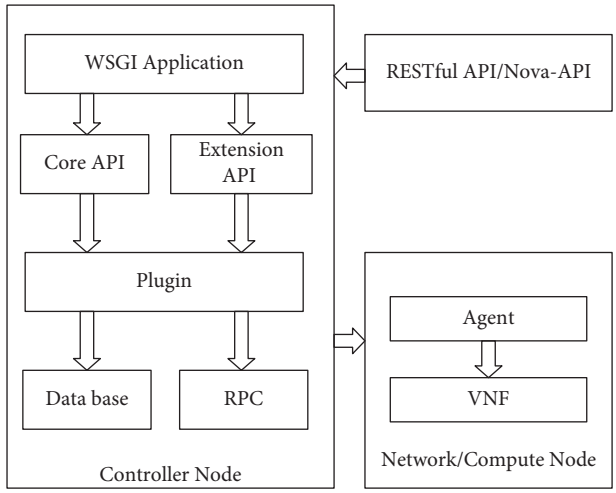

Figure 5: Hierarchical trusted invocation association.

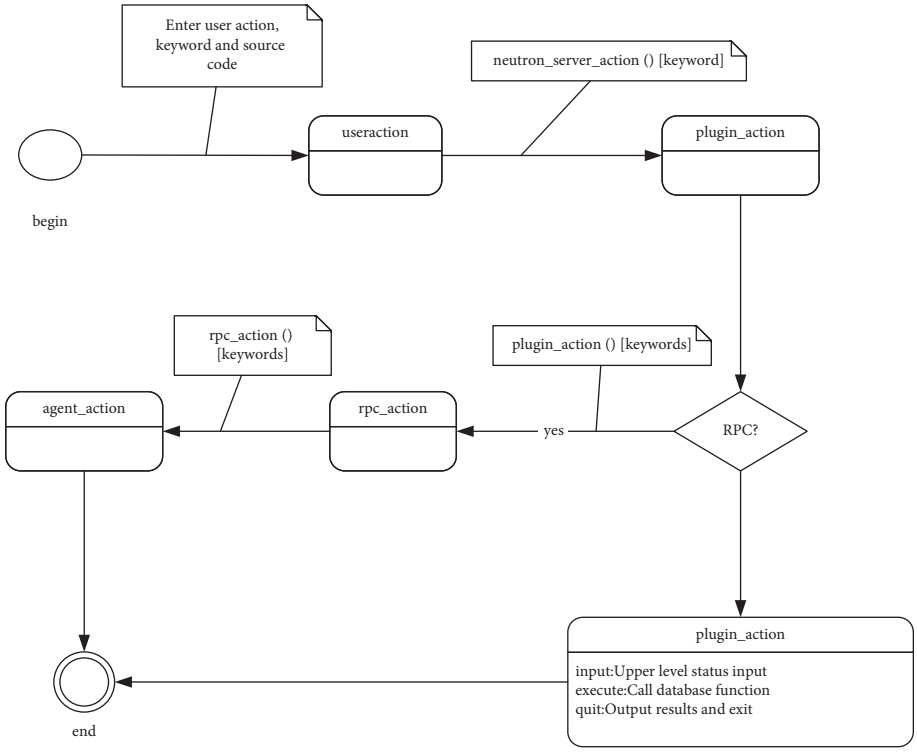

Figure 6: The trusted behavior description based on finite state machine for tenant's virtual network action operation.

is obtained, it will be traversed and matched with related to operation function keywords in the keyword databases from various levels of cloud service. And then the corresponding behavior state can be created through hierarchical trusted invocation association. When traversed to the bottom level, the end state constitutes the trusted behavior of a legitimate user. Finally, the generated tenant normal behavior model is stored in a trusted third-party module to prevent internal malicious actors from destroying it. The specific algorithm is shown in Table 1.

Let's take tenant's create network operation *create_network* as an example to illustrate the process of building trust behavior. As shown in Figure 7, the specific steps are as follows.

(i) Create the root node in the user layer and enter the resource file to get the action *create_network*.

(ii) Enter the base file in the virtual network control layer, find the corresponding interface function according to the obtained passed parameter *create_network*, and pass the obtained interface function to the lower layer as the parameter.

(iii) The management process enters the management process document at the virtual network layer, finds the corresponding interface function according to the top of the incoming *create_network* () function, and judges whether there

Table 1: Trusted behavior generation algorithm.

```
INPUT : User action, Keywords, Network service component source code
OUTPUT : Normal trusted behavior
Func createNormalBehavior(action string, keywords []string, soure code * file){
        begin_state: = action
        normal_node: = begin_state
      for action! = nil{
        if next_state = next(begin_state) {//read next level
      normal_node = append(node, next_state)
    }
  } return normal_node
}
```

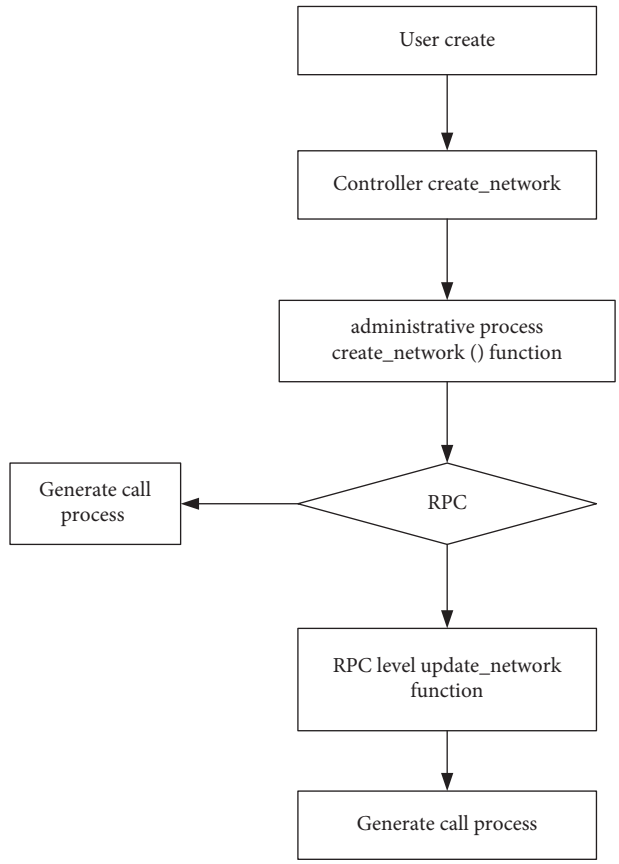

Figure 7: A trusted behavior generation instance on create network.

is any RPC behavior, therefore, generates choice node in this layer. If there is no RPC behavior, a complete calling process is generated; otherwise, the interface function of this layer is passed down as a parameter to the next layer.

(iv) Enter the RPC file in the RPC layer, find the corresponding interface function *update_network* (), and pass the obtained interface function as a parameter to the lower layer.

(v) Enter the service process file in the service process layer, find the corresponding interface function through the parameters passed in the upper layer, and the agent creates the calling process and returns for the last layer of behavior.

With the development of cloud virtual network services, users' management requirements for virtual network may be change. When a user has a new operation to his virtual network, the user's trusted behavior can be updated based on the previous source code analysis and log information based on actual operations.

4.3. Actual Behavior Tracing. In VNGuarder, collection points are set up at all levels of virtual network service. For example, the remote call interface is added with the recorded time and the relevant interface information of the output operation. When the corresponding invocation operation occurs in each collection point, the relevant interface or function information and the calling time are printed into a log, that is, *log.* (*time: API/func*). Normally, the log

information will be output to network service log of controller node, and management implementation log of network node and compute node in cloud platform. At the same time, the change of virtual network can be found in time through the deployment of real-time monitoring module. Once the virtual network configuration changes, the behavior information of each interface invoked is logged and then the actual behavior can be traced in order of time. That means

$$\text{Actions} = \left\{\frac{\text{time1: apil}}{\text{func1}}\right\}, \left\{\frac{\text{time2: api2}}{\text{func2}}\right\}, \left\{\frac{\text{time13: api3}}{\text{func3}}\right\}, \ldots\ldots \left\{\frac{\text{timen : apin}}{\text{funcn}}\right\}. \quad (2)$$

The trace-enable algorithm is shown in Table 2.

4.4. Behavior Matching. Comparing the above traced-enable actual behavior with the results of hierarchical trusted invocation association, the current request can be considered as a normal request initiated by a legitimate user if a complete match can be made. If there is not an exact match, then it should be an unauthorized request, which may be malicious attacks from insiders. In addition, although malicious behavior can be triggered from any node in cloud platform, it will eventually be executed at the bottom. Therefore, in order to optimize the matching efficiency, behavior matching should start from the bottom. The insider threat detection algorithm based on behavior matching is shown in Table 3.

5. Evaluation

5.1. Experimental Environment. VNGuarder has been deployed in Openstack cloud platform and the specific implementation architecture is shown in Figure 8. In the experiment, three physical machines with I5 7400 CPU, 8G memory, and Ubuntu operating system were used. One of them acts as a controller node, while the other two act as network and compute nodes, respectively. The above-mentioned hierarchical trusted invocation association scheme and the proposed FSM algorithm are partially deployed to the controller node to generate the complete trusted behavior of a legitimate tenant as a matching object. At the same time, the trace-enable mechanism is deployed to the controller node, network node, and compute node to realize real-time monitoring to neutron agents, where collection points are set on the interfaces of neutron-server, neutron-plugin, RPC remote procedure call, neutron-agent management implementation interface, and virtualization management interface, etc. When real-time monitoring module detects related operations, each collection point will output the corresponding action and its time information. Finally, according to the information collected at each collection point, the behavior matching scheme is adopted to identify the behavior of malicious insiders by comparing the traced behavior with the previous normal behavior of the legitimate tenant. The interaction diagram is shown in Figure 9.

5.2. Validity Check. In this section, three groups of experiments are conducted to verify the validity of VNGuarder, when a malicious cloud insider is engaged in illegal call from different nodes and perform different malicious operations.

First, an experiment was performed to verify that malicious changes made by the malicious manager to the user's virtual machine port resource, such as changing the IP address of virtual interface, can be detected. In the experiment, we implement malicious operations by directly changing virtual devices in compute nodes. Under the OpenStack platform, the carrier body of port is realized by *ovs*. The *ovs agent* will receive the message from *neutron-server* and then configure the port, when the port changes. We configure the *ovs agent* such as adding ports to the original network by using the command "*ovs-vsctl add-port br0 eth*". As shown in Figure 10, after VNGuarder has been deployed, the actual invocation behavior is built by trace-enable mechanism, if the monitoring module dynamically monitors the port invocation occurred at some point in real time. When the behavior matching module completes the comparison between the actual invocation behavior and the hierarchy trusted invocation, it feedbacks the final detection result.

Second, an experiment was performed to verify that malicious configuration by the cloud operator to the user's subnet resources, such as changing the IP scope of the user's network. In the experiment, in order to achieve a malicious operation different from the first experiment, the neutron-plugin process of controller node is directly called by passing parameters for the plugin function, and then the corresponding driver and RPC process are called to realize the malicious configuration of subnet. As shown in Figure 10, the monitoring module of VNGuarder reenters real-time monitoring mode after having completed the first monitoring. It can detect and reflect the subnet's invocation behavior.

Finally, an experiment was performed to verify that the destruction of router resource by the cloud operator, such as adding ports to the router. Compared with the previous experiment, the neutron-server process is invoked in this experiment. A malicious insider can call the function in the controller class in the neutron-server, pass parameters by using the function, and then continue to call the plugin process, RPC process, and agent process to achieve the purpose of changing the router. As shown in Figure 10, the monitoring module in VNGuarder has been working. Whenever the router invocation behavior occurs, the VNGuarder appproach can detect and report back.

5.3. Detection Rate. Repeated experiments were carried out on the three typical attack behaviors mentioned above and normal operation behaviors of users. Statistical analysis was performed on whether the malicious change operations of

Table 2: Trace-enable algorithm.

```
Input: log file, real-time monitoring information output : Current
operation correlation information
  func trance_back(log * file, time * file){
    for log! = nil{
      first_node,err: = traverse(log_i,time)
      if err! = nil{
        Break
    }else{
        Virtual_node: = append(node, first_node)
        next_node: = traverse(log_i,time)
      }
    } return virtual_node
}
```

Table 3: Behavior matching algorithm.

```
INPUT:virtual_node, normal_node output : Boolean value
Func match(virtual_node, normal_node){
flag: = 0
If virtual_node[:-1]! = nil{
If virtual_node[:-1] = = normal_node[:-1]{
For virtual_node.next! = normal_node.next{
flag = 1
Break
} if flag = = 1{
Return false
}Else{ return true
}
}
```

port resource, subnet resource, router resource, and the normal operation of users could be correctly monitored after the deployment of VNGuarder. As shown in Figure 11, the attacks caused by the above three typical operations can be correctly monitored more than 90%, and there are fewer misjudgments for normal user's operations with VNGuarder.

5.4. Impact on Virtual Network Creation Time. In VNGuarder, the trace-enable mechanism extracts process information and generates a log file, which provides some performance for virtual machine runs and virtual network creation. In the experiment, we have counted network creation time with and without VNGuarder deployment. We created a virtual network for a tenant in the cloud environment, and recorded the time from the start of creating a virtual machine to the completion of the virtual network as the virtual network creation time, and conducted the experiment 10 times. As shown in Figure 12, where the horizontal axis represents the number of experimental rounds, and the vertical axis represents the time to set up a virtual network. The results show that VNGuarder deployment has little effect on the virtual network creation time.

5.5. CPU and Memory Performance Loss. VNGuarder is deployed on different cloud hosts and only responds when the virtual network is configured. Therefore, the cloud host performance, such as CPU usage and memory usage, is affected only when the virtual network configuration is performed. Otherwise, the cloud host performance is not significantly affected. At the same time, VNGuarder only monitors the working nodes in real-time which can reduce unnecessary CPU and memory usage. In this experiment, some performance detection tools are used to conduct statistical analysis of CPU and memory consumption and compare the performance changes before and after deployment of VNGuarder. The benchmark is calculated as follows.

As shown in Figure 13 and Figure 14, the horizontal axis represents the experiment times, the vertical axis represents the average decline rate of CPU or memory which reflects the performance changes without network operation and with the network configuration after VNGuarder has been deployed. The results show that the virtual machine is operated normally and the CPU or memory performance is basically unaffected when the network configuration operation is not involved. When the network configuration operation exists, the performance overhead will change significantly, but the range of change is still within the acceptable range.

5.6. Functional Comparison with the Existing Work. In this subsection, we compare VNGuarder with the existing related work [27–30], including the internal threat points of cloud system that different solutions focus on, whether they can monitor single node or multiple nodes of cloud platform, and whether they support behavior association tracking. The Functional Comparison shown in Table 4.

6. Discussion

At present, machine learning methodologies are introduced for mitigating insider threats [31], where multiple data sources are used to find associations. For example, machine learning can analyse whether all intrarelational behavior is malicious based on the behavior of insiders who have already been flagged as at risk. However, this approach can produce false positives because internal staff are often assigned new permissions when they are assigned new business. Therefore, this kind of methods needs to increase data continuously to improve the accuracy of the training model algorithm. In contrast to the uncertainties associated with machine learning methods, VNGuarder can detect internal threat without changing the hierarchy trusted behavior model based on individual permissions.

In our early work [21], we focus on the protection of data in cloud compute nodes, through the use of behavior trace and access control means to protect the data in a computing node, although this method can effectively avoid the internal malicious personnel directly to the protection of data storage computing node but ignore the internal management of malicious or operations staff through the virtual network malicious way destroy the tenant management network, and then get the user data.

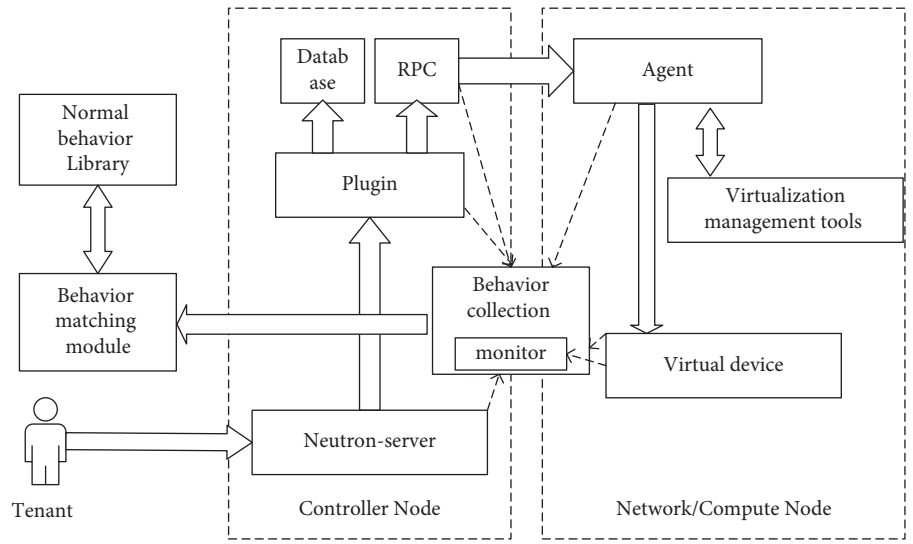

Figure 8: VNGuarder deployment in Openstack cloud platform.

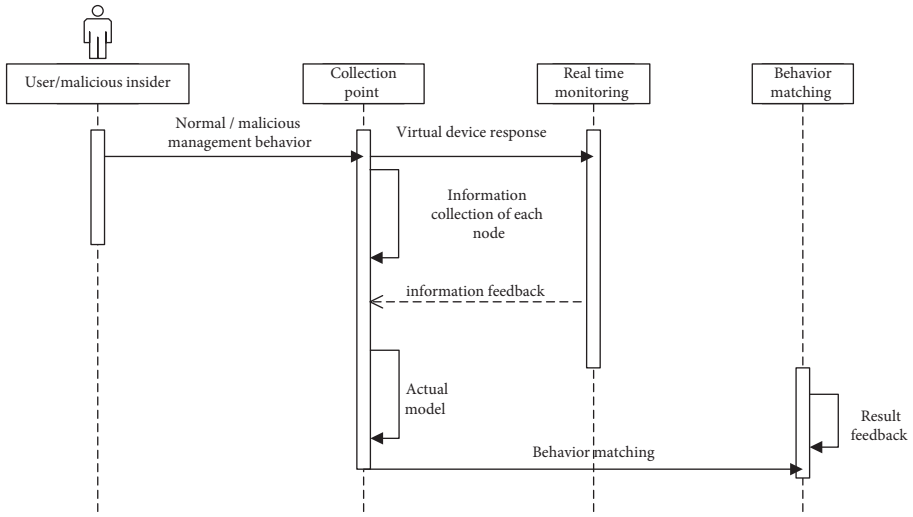

Figure 9: The interaction diagram in VNGuarder.

```
openstack@openstack:~$ ./VNGuarder_monitor -d 2021-05-20 -t 10:06:25 -v port
the time:2021-05-20 10:06:25,normal operation on port
openstack@openstack:~$ ./VNGuarder_monitor -d 2021-05-20 -t 11:32:30 -v port
the time:2021-05-20 11:32:30,abnormal and illegal operation on port
openstack@openstack:~$ ./VNGuarder_monitor -d 2021-05-20 -t 11:32:30 -v port
the time:2021-05-20 11:32:30,abnormal and illegal operation on port
```

Figure 10: The effectiveness verification of VNGuarder in monitoring resource changes such as port, subnet, and router.

Therefore, finding and blocking virtual network malicious management and configuration has become a key to improve the overall prevention of internal malicious threats in cloud computing environment.

At the same time, VNGuarder can effectively reduce the number of threats judged by the system because it compares and determines whether there is a threat in the trusted hierarchical model only after the collection point has collected the corresponding information. In the traditional way, after the data is obtained, the threat judgment is completed through continuous screening. Therefore, VNGuarder can effectively reduce the complexity in practical application.

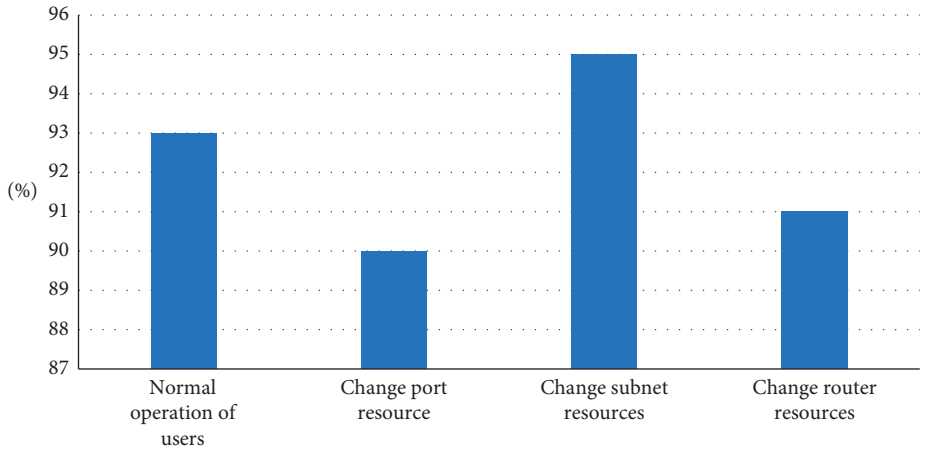

Figure 11: Detection rate under different Network management operations with VNGuarder.

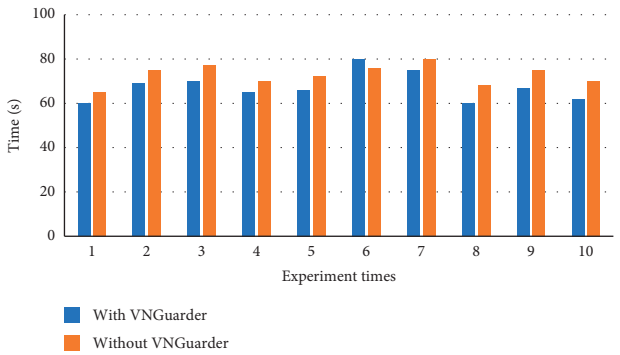

Figure 12: Virtual network creation time in 10 rounds under OpenStack with VNGuarder and without VNGuarder.

CPU = LCPU − PreCPU Memory = LMemory-PreMemory
* *CPU= Change in utilization rate of CPU* * *LCPU = After VNGuarder of CPU* * *PreCPU = Per VNGuarder of CPU* * *Memory = Change in utilization rate of Memery* * *LMemory = After VNGuarder of Memery* * *PreMemory = Per VNGuarder of Memery*

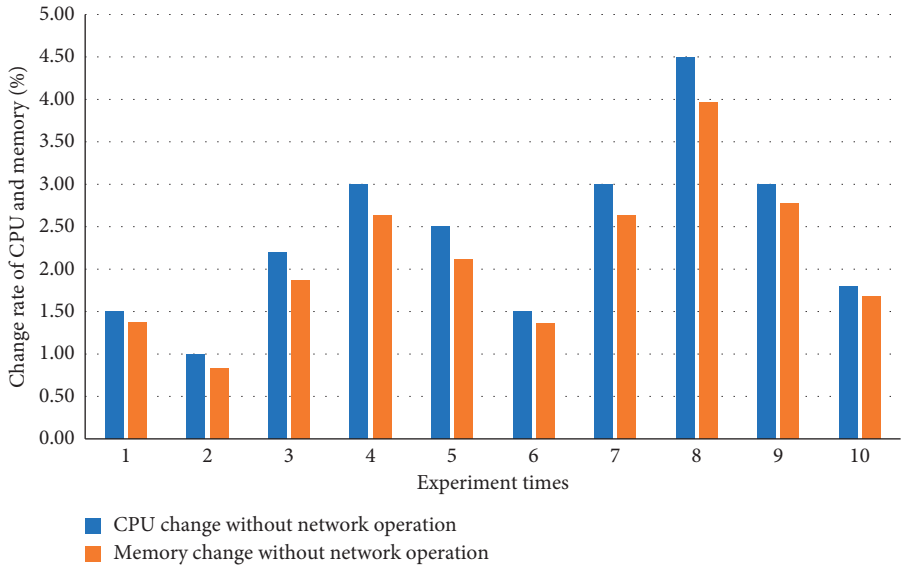

Figure 13: Performance change of CPU and Memory in 10 rounds under Openstack without network operation.

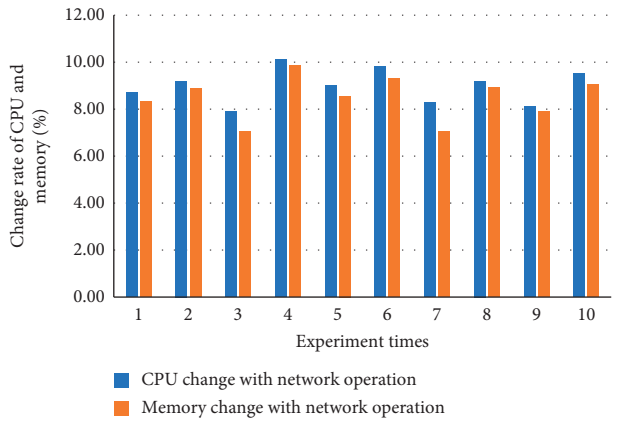

Figure 14: Performance change of CPU and Memory in 10 rounds under Openstack with network operation.

Table 4: Functional comparison.

Method	Internal threat point	Monitor single node	Monitor multinode	Behavior association tracking
[27]	VM service	Yes	No	No
[28]	VM image	Yes	Yes	No
[29]	Tenants and managers	Yes	No	No
[30]	Tenants and managers	Yes	No	No
VNGuarder	Virtual network service	Yes	Yes	Yes

7. Conclusions and Future Work

Virtual network security in the cloud environment will eventually affect on edge-assisted Internet of things applications. There is little research on insider threat attack to virtual network management. This paper proposes an approach named VNGuarder to protect IaaS users' private networks from being maliciously changed or destroyed by insider. In VNGuarder, a trace-enable mechanism is introduced to integrate real-time monitoring and log analysis. A hierarchical trusted invocation association method based on tenants' normal behavior is proposed to provide a basis for discovering illegal invocation of insiders to virtualization management tools and virtual network-related processes. VNGuarder has been implemented on OpenStack platform, and the effectiveness and performance evaluation have been verified by experiments. The results show that VNGuarder can identify several important insider threats like illegal management and configuration modification to virtual network device through virtualization tools with a small performance overhead.

The ongoing work is twofold. On the one hand, the normal hierarchical trusted invocation behavior is constructed based on the source analysis of OpenStack Neutron

service. However, with the continuous iteration of Neutron service, the source code logic structure may change. We plan to design a trusted invocation behavior modelling method based on machine learning algorithm, in order to improve the efficiency and adaptability of trusted behavior construction. On the other hand, VNGuarder can find a malicious insiders' behavior only when the insider threat is triggered. Hence, a tenant's private network may be exposed under malicious threats for a period of time before he receives attack detection feedback. In this case, we need to further fine-grain and classify the management operations of virtual network, including determining which management interface cloud managers can invoke and which cannot. For the latter, an effective access control mechanism should be designed to control access to reduce the risk of random management and misoperation by malicious insiders.

Data Availability

All data used to support the findings of this study are included within the article.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported in part by the National Science Foundation of China under Grant 61502017 in part by the Natural Science Foundation of Beijing Municipality under Grant M21039

References

[1] C. Wang, D. Wang, G. Xu, and D. He, "Efficient privacy-preserving user authentication scheme with forward secrecy for industry 4.0," *Science China Information Sciences*, vol. 65, no. 1, Article ID 112301, 2022.

[2] W. Wang, H. Xu, M. Alazab, T. R. Gadekallu, Z. Han, and C. Su, "Blockchain-based reliable and efficient certificateless signature for IIoT devices," *IEEE Transactions on Industrial Informatics*, pp. 1551–3203, 2021.

[3] W. Li, W. Meng, C. Su, and L. F. Kwok, "Towards false alarm reduction using fuzzy if-then rules for medical cyber physical systems," *IEEE Access*, vol. 6, pp. 6530–6539, 2018.

[4] S. Qiu, D. Wang, G. Xu, and S. Kumari, "Practical and provably secure three-factor Authentication protocol based on extended chaotic-maps for mobile lightweight devices," *IEEE Transactions on Dependable and Secure Computing*, vol. 17, no. 3, 2020.

[5] W. Wang, H. Huang, L. Zhang, and C. Su, "Secure and efficient mutual authentication protocol for smart grid under blockchain," *Peer-to-Peer Networking and Applications*, vol. 14, no. 5, pp. 2681–2693, 2021.

[6] Z. Lian, W. Wang, and C. Su, "COFEL: communication-efficient and optimized federated learning with local differential privacy," in *Proceedings of the ICC 2021-IEEE International Conference on Communications*, pp. 1–6, IEEE, Montreal, Canada, June 2021.

[7] T. Wang, Y. Quan, X. S. Shen, T. R. Gadekallu, W. Wang, and K. Dev, "A privacy-enhanced retrieval technology for the cloud-assisted Internet of things," *IEEE Transactions on Industrial Informatics*1 page, 2021.

[8] "Open Stack Networking architecture-Security Guide documentation (openstack.org)," 2021, https://docs.openstack.org/security-guide/networking/architecture.html.

[9] L. Rui, Y. Jiawei, H. Dongjie, and C. Hua, "On virtual network access based on open VSwitch," *Computer Applications and Software*, vol. 31, no. 5, pp. 308–311, 2014.

[10] P. Goyal and A. Goyal, "Comparative study of two most popular packet sniffing tools-Tcpdump and Wireshark," in *Proceedings of the 2017 9th International Conference on Computational Intelligence and Communication Networks (CICN)*, pp. 77–81, Girne, Northern Cyprus, September 2011.

[11] M. R. Sutradhar, N. Sultana, H. Dey, and H. Arif, "A new version of kerberos authentication protocol using ECC and threshold cryptography for cloud security," in *Proceedings of the 2018 Joint 7th International Conference on Informatics, Electronics & Vision (ICIEV) and 2018 2nd International Conference on Imaging, Vision & Pattern Recognition (icIVPR)*, pp. 239–244, Kitakyushu, Japan, June 2018.

[12] B. Wang, Y. Zheng, W. Lou, and Y. T. Hou, "DDoS attack protection in the era of cloud computing and Software-Defined Networking," *Computer Networks*, vol. 81, pp. 308–319, 2015.

[13] F. Sabahi, "Cloud computing security threats and responses," in *Proceedings of the 2011 IEEE 3rd International Conference on Communication Software and Networks*, pp. 245–249, Xi'an, China, May 2011.

[14] N. Agrawal and S. Tapaswi, "Low rate cloud DDoS attack defense method based on power spectral density analysis," *Information Processing Letters*, vol. 138, pp. 44–50, 2018.

[15] M. O. Alassafi, R. AlGhamdi, A. Alshdadi, A. Al Abdulwahid, and S. T. Bakhsh, "Determining factors pertaining to cloud security adoption framework in government organizations: an exploratory study," *IEEE Access*, vol. 7, pp. 136822–136835, 2019.

[16] J. Tang, Y. Cui, Q. Li, K. Ren, J. Liu, and R. Buyya, "Ensuring security and privacy preservation for cloud data services," *ACM Computing Surveys*, vol. 49, no. 1, pp. 1–39, 2017.

[17] A. Pandey and S. Srivastava, "An approach for virtual machine image security," in *Proceedings of the 2014 International Conference on Signal Propagation and Computer Technology (ICSPCT 2014)*, pp. 616–623, Ajmer, India, July 2014.

[18] S. Baik, Y. Lim, J. Kim, and Y. Lee, "Adaptive flow monitoring in SDN architecture," in *Proceedings of the 2015 17th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pp. 468–470, Busan, Korea (South), August 2015.

[19] K. Zhang, H. F. Wang, Y. C. Ma, and Z. Li, "The status and development trends of virtual private network technology," *Key Engineering Materials*, vol. 474-476, pp. 79–82, 2011.

[20] T. Haibo Chen and H. Chen, "FlexCore: dynamic virtual machine scheduling using VCPU ballooning," *Tsinghua Science and Technology*, vol. 20, no. 1, pp. 7–16, 2015.

[21] L. Lin, S. Li, B. Li, J. Zhan, and Y. Zhao, "TVGuarder: a trace-enable virtualization protection framework against insider threats for IaaS environments," *International Journal of Grid and High-Performance Computing (IJGHPC)*, vol. 8, no. 4, pp. 1–20, 2016.

[22] A. Hussein, I. H. Elhajj, A. Chehab, and A. Kayssi, "SDN security plane: an architecture for resilient security services," in *Proceedings of the 2016 IEEE International Conference on Cloud Engineering Workshop (IC2EW)*, pp. 54–59, Berlin, Germany, April 2016.

[23] Y. F. Shao and Z. Jia, "Research on software defined network security technology," *Radio Engineering*, vol. 46, no. 4, pp. 13–17, 2016.

[24] M. B. de Carvalho, R. P. Esteves, G. da Cunha Rodrigues, L. Z. Granville, and L. M. R. Tarouco, "A cloud monitoring framework for self-configured monitoring slices based on multiple tools," in *Proceedings of the 9th International Conference on Network and Service Management (CNSM 2013)*, pp. 180–184, Zurich, Switzerland, October 2013.

[25] Csa Top Threats Working Group, "The treacherous twelve cloud computing top threats in 2016," 2016, https://cloudsecurityalliance.org/artifacts/the-treacherous-twelve-cloud-computing-top-threats-in-2016/.

[26] J. Bouche and M. Kappes, "Attacking the cloud from an insider perspective," in *Proceedings of the 2015 Internet Technologies and Applications (ITA)*, pp. 175–180, Wrexham, UK, September 2015.

[27] Z. Yu, W. Zhang, and H. Dai, "A trusted architecture for virtual machines on cloud servers with trusted platform module and certificate authority," *Journal of Signal Processing Systems*, vol. 86, no. 2-3, pp. 327–336, 2017.

[28] R. K. Hussein, A. Alenezi, H. F. Atlam, M. Q. Mohammed, R. J. Walters, and G. B. Wills, "Toward confirming a framework for securing the virtual machine image in cloud computing," *Advances in Science, Technology and Engineering Systems Journal*, vol. 2, no. 4, pp. 44–50, 2017.

[29] V. Kansal and M. Dave, "Proactive DDoS attack detection and isolation," in *Proceedings of the 2017 International Conference on Computer, Communications and Electronics (Comptelix)*, pp. 334–338, Jaipur, India, July 2017.

[30] M. Singh, B. M. Mehtre, and S. Sangeetha, "Insider threat detection based on user behavior analysis," in *Machine Learning, Image Processing, Network Security and Data Sciences. MIND 2020. Communications in Computer and Information Science*, A. Bhattacharjee, S. Borgohain, B. Soni, G. Verma, and XZ. Gao, Eds., vol. 1241pp. 559–574, 2020.

[31] Y. Xin, L. Kong, Z. Liu et al., "Machine learning and deep learning methods for cybersecurity," *IEEE Access*, vol. 6, pp. 35365–35381, 2018.