2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)

BTDetect: An Insider Threats Detection Approach Based on Behavior Traceability for IaaS Environments

Li LIN[1,2], Shuang LI[1], Xuhui LV[1], Bo LI[3]
[1]College of Computer Science, Dept. of Information Technology, Beijing University of Technology, Beijing. China
[2]Beijing Key Laboratory of Trusted Computing, Beijing, China
[3] State Key Laboratory of Software Development Environment, Beihang University, Beijing, China.
Email: linli_2009@bjut.edu.cn

***Abstract*—How to detect malicious insiders' improper access to tenant data has become more crucial in IaaS cloud environment, especially with the cloud administrators gaining more control on customers' virtual machines and data in reality. In this paper, we propose an insider threats detection approach based on behavior traceability called BTDetect. First, we analyze the service invocation interfaces of IaaS cloud environment, such as computing service, remote call, management implementation and virtualization management, and condense the complete process of cloud user behavior. Using tree-based modeling technique, a behavior-tree construction algorithm is proposed to construct the normal behavior tree that can describe various legal operations of cloud users. Second, we set up trace points of cloud service behavior on multi-layer cloud service APIs, then we collect information of each interface being invoked across multiple nodes. Third, we use underlying virtualization behavior keyword matching technology to match the collected behaviors with the user's normal behavior tree and then the malicious internal threat can be identified through tree-based integrity analysis. Finally, some experiments are conducted to evaluate the feasibility and veracity of the proposed method in Openstack platform. The results suggest that our method can not only identify internal threat but also have high recognition rate.**

Keywords—IaaS cloud security, virtualization, insider threats, behavior traceability

I. INTRODUCTION

As one of the network-enabled technologies, cloud computing has been broadly adopted in the industry through numerous cloud service models [1]. According to Gartner Forecast, the global cloud computing service market size will reach 354.6 billion us dollars in 2022 [2]. Existing cloud service models can be divided into three types: *Infrastructure as a service* (IaaS), *platform as a service* (PaaS), and *software as a service* (SaaS). IaaS usually provide users with the required IT resources such as processors, memory, disks, networks and so on by providing independent virtual machines. Users only need to configure and install the operating system and upper applications on the virtual machines.

Although cloud computing provides convenient computing and storage services to tenants, it inevitably suffers the threat from the Internet and some important new security issues are also born [3]. Among the IaaS service, the virtual machine server used by the tenant stores the data in the form of mirror files, while the internal staff of the cloud service has complete control over the tenant's data. Some of the recent security vulnerabilities in the cloud environment are due to internal negligence. For example, the bank information of 2 million vodafone customers was attacked and leaked by insiders [4] and the infamous NSA data theft by Edward snowden. In both cases, insiders stole data after gaining access to the data.

While cloud service providers (CSP) typically make privacy commitments to cloud users, there is still a risk that their data will be compromised or corrupted. Recent research has focused on protecting user data or virtual machines from unreliable CSPs. Ali [5] proposed a secure data storage and sharing method based on cloud environment, which uses trusted third parties to protect user data and establishes a series of security measures from data storage to data transmission. However, this will face the problem of the transmission speed of user data between cloud environment, which constrain the performance of the original efficient cloud environment.

Kazim [6], Pandey [7] focused on the risk that the virtual machine images of cloud users might be leaked after being uploaded to the cloud environment. They proposed an AES encryption method to store the encryption of user image to the cloud platform, and then decrypt each time to complete the protection of the image data. This method has a high time complexity of encryption algorithm. With the increase of data volume, user image will also get larger and larger, which will greatly affect the virtualization performance of the cloud environment. Tan [8], Xia [9], and Miu [10] use nested virtualization to monitor the behavior of CSPs and protect the security of virtual machines from tampering with malicious CSPs. However, it needs to distinguish VMM-level operations from user-level operations and requires a lot of extra work on current cloud platform implementation.

The above researches assume that CSPs are not credible, but in fact CSPs have no intention to disclose user data intentionally, and the real enemy that threatens the security of cloud users is internal attack. According to the report from the Cloud Security Alliance, malicious internal attacks are listed as one of the most important threats to cloud computing [3]. However, detecting internal attacks is not an easy task. Firstly, the characteristics of cloud services invoked by malicious insiders and cloud services invoked by cloud users are indistinguishable. Secondly, even if some malicious behavior features are obtained, there is no guarantee that malicious actions other than these will not occur.

978-0-7381-2646-3/21/$31.00 ©2021 IEEE

DOI 10.1109/ISPA-BDCloud-SocialCom-SustainCom52081.2021.00055

Finally, if we define normal behavior, we need to analyze the normal behavior of the entire IaaS cloud service. Therefore, it is actually necessary to propose some new behavior collection and security detection technologies to detect the security of cloud user data.

To solve the above problems, we propose an insider threats detection approach based on behavior traceability called BTDetect. Since the user data in the IaaS cloud environment is mostly hosted to the cloud by the virtual machine image file, the main concern in this paper is the detection of the access behavior of the virtual machine image file. Firstly, we analyze the service invocation interfaces of IaaS cloud environment, such as computing service, remote call, management implementation and virtualization management, and condense the complete process of cloud user behavior. Using tree-based modeling technique, a behavior-tree construction algorithm is proposed to construct the normal behavior tree that can describe various legal operations of cloud users. Secondly, we set up trace points of cloud service behavior on multi-layer cloud service APIs, then we collect information of each interface being invoked across multiple nodes. Thirdly, we use underlying virtualization behavior keyword matching technology to match the collected behaviors with the user's normal behavior tree and then identify malicious internal threat through tree-based integrity analysis.

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

- Considering that the existing machine learning-based methods need to utilize known threat behavior libraries and cannot identify unknown threats, this paper adopts the idea of identifying malicious unknown internal threats based on normal behavior, and proposes behavior tree construction method based on API association analysis. Through the association analysis of the IaaS cloud service call interface and related source code, a normal behavior tree for the user's legal operation is constructed.
- The existing cloud security auditing method only audits the interface behavior of a cloud service, and cannot trace the data access behavior of the cloud service user across multiple nodes, and it is difficult to determine whether the current data access request is issued by the user. This paper considers the situation of malicious calls across multiple nodes and collects behavior information that each interface is called across multiple nodes. The collected information is matched with the behavior of the previously constructed normal behavior tree to identify malicious threats through tree-based integrity analysis.
- We have implemented BTDetect on the Openstack cloud platform and conducted several comprehensive experiments to evaluate its validity and accuracy. The experimental results show that BTDetect can not only identify internal threat but also have high recognition rate.

The rest of the paper is organized as follows. Section II introduces the related work. Section III introduces the system model. Section IV introduces BTDetect in detail. The implementation of BTDetect in Openstack platform and the experimental results are given in Section V. Finally, we conclude the paper in Section VI.

II. RELATED WORK

With the development of computer hardware [23-25], software [26-27], and networks [28-29], security and privacy[30-32] become a more and more important issue in various applications. For example, in health care [33] and autonomous drive [34], the security and privacy had been put as the first priority in system design [35]. With the advance in big data and cloud computing [36-37], data security [38-39] appears to be a new hot research area. The current data security issues for cloud users are generated by the outsourcing capabilities of cloud services. Tenants lose absolute control over these resources. Some malicious internals can use their permissions to access and even tamper with user data and private information.

Paul [11] designed an internal threat detection method based on malicious behavior, which provides the idea of behavior based internal threat detection for this paper. Lou [12] detected the resources occupied by service based on monitoring devices usage. However, monitoring resources of computing, storage and network can only aware the service exceptions but not identify the malicious behaviors of insider. Gupta [13] proposed various security risks and vulnerabilities in the cloud environment under virtualization technology, and mentioned that the data of cloud user is facing the risk of unauthorized access and tampering.

Zhang [14] proposed to use CloudMonatt to improve the security level of the cloud environment by verificating of the security functions provided by the client VM's cloud virtual machine. But this method can only be applied to the running virtual machine and it cannot monitor the virtual machine instance image file. Zhou [15] first proposed a behavior tree based on the system call interceptor and tested the virtual machine behavior based on the trusted technology. Cheh [16] proposed a method of access logs behavior detection, which uses the defined Markov model to detect low score behavior by input the user behavior log. However, the internal threats in the cloud environment can be realized through malicious API calls with higher permissions. The behavior generated by these threats is hard to be defined in advance with Markov model.

In the real IaaS cloud environment, malicious insiders use their own permissions to invoke different node cloud service interfaces to perform indirect illegal operations on user data. Therefore, some scholars have proposed cloud security audit technology to solve the above problems. Wang [17] elaborated the research progress and shortcomings of current cloud security audit in three aspects: log auditing, cloud storage auditing and configuration auditing. Masetic [18], Bhamare [19] and Aldairi [20] proposed the use of machine learning algorithms to detect threats in cloud environments by analyzing threat models and cloud environment architectures in cloud environments. Mishra [21] used a machine learning decision tree algorithm to classify a large number of virtual machine malicious system calls, obtained malicious behavior related system calls and completed classification detection based on the characteristics of malicious behavior.

The above work shows that threat auditing based on machine learning can detect some known threats, but still need to constantly update threat model library to support the detection of unknown threats. Tian [22] introduced a trusted third party to verify whether the operation behavior log from a cloud user is abnormal, but it only performed security auditing and detection on a single node in the cloud environment and could not accurately distinguish whether a legitimate access request from a cloud user or a malicious insider is illegal when an internal person invokes multiple different node services.

III. SYSTEM MODEL

The process of a normal cloud user using the IaaS cloud service from the management interface to the user virtual machine image is as shown in Fig.1. The cloud user needs to obtain the authentication and authorization component such as keystone when calling the cloud user service interface. After the authentication is obtained, the cloud service interface can be called normally. The malicious internal staff have high authority and can indirectly operate the user image by calling some interfaces in the calling process, thereby posing a threat to the user image data file. For example, attackers can first invoke a virtualization service process such as qemu-kvm at a compute node to access a user image file and then they enter the virtual machine system by starting the qemu-kvm process, or call qemu-img to modify the image file. Secondly, attackers can invoke a virtualization management process interface such as libvirt to manipulate or monitor the running state of virtual machine. Finally, attackers can invoke the computing service management interface in the control component. According to the ID of the user VM, they invoke related services such as power-on, power-off, and suspend from the computing service management interface to control the running state of virtual machine. After that, they can indirectly access the image file through calls from different levels of service interfaces to bring data threats.

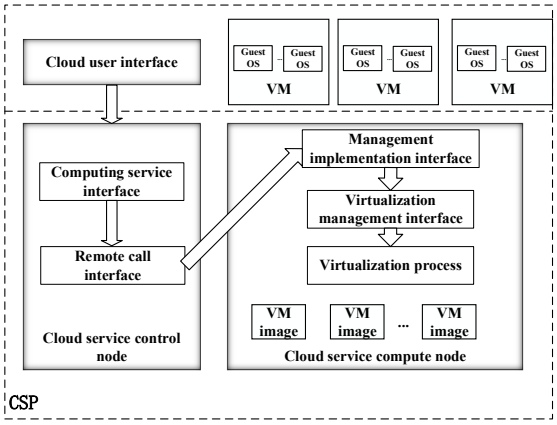

Fig.1. IaaS cloud service interface invocation process

The objective of this paper is to propose an internal threat detection method based on behavior traceability called BTDetect. The method can detect those malicious behaviors where malicious insiders invoke various hierarchical cloud services to access or tamper with users' data.

IV. DESIGN OF BTDETECT

A. Working Principles

As shown in Fig.2, there are three modules introduced in BTDetect, which are normal behavior tree building module, behavior acquisition module and behavior detection module.

The normal behavior tree building module adopts a behavior tree construction method based on multi-layer API association analysis. It establishes different types of logical nodes according to the calling relationship between different nodes. From the user entry node to the last implementation node, the behavior of each level in the cloud service corresponds to the node in the behavior tree. Then a normal behavior tree library is provided for subsequent internal threat discovery. Taking an open-source cloud platform Openstack as an example, a behavior tree represents different operational levels of current cloud service interface, including a computing service (e.g, Nova) interface, and virtualization management (e.g, libvirt) interface etc.

Each level has its own independent user's behavior using virtual machine service-related calls, which are the basic nodes of the behavior tree. Before using the algorithm to analyze the relevant source code, we need to define the keywords for each layer of behavior-related source code. If there is a keyword, the node is created according to the nature and added to the behavior tree. The specific process is shown in Fig.3.

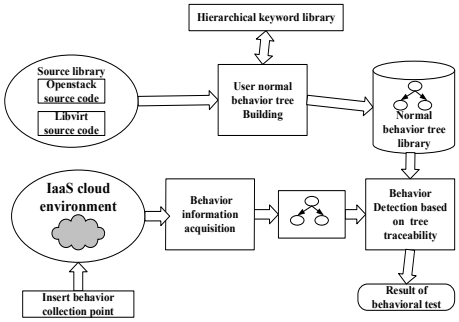

Fig.2. Architecture of BTDetect

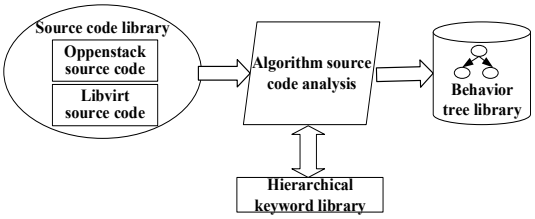

Fig.3. Behavior tree building process

The behavior acquisition module mainly sets the traceability point of the cloud service behavior on the multi-layer APIs such as the computing service interface, the remote calling interface, the management implementation interface and the virtualization management interface. A malicious insider can threaten an image file by calling a computing service interface, a management implementation interface, a virtualization management interface and a virtualization process. Hence, behavior collection points are introduced at these interfaces or processes. The behavior records can be obtained and collected according to time and behavior type when the relevant service interface is called, as shown in Fig.4.

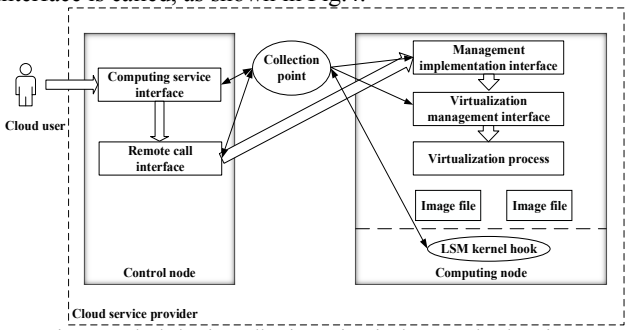

Fig.4. Set the behavior collection points in the IaaS cloud environment

Based on the information collected by each traceability point, the behavior detection module uses the behavior keyword matching technology to track and match the behavior in the user's normal behavior tree constructed earlier. Malicious internal threats are identified through tree-based integrity analysis. The internal threat detection process is shown in Fig.5.

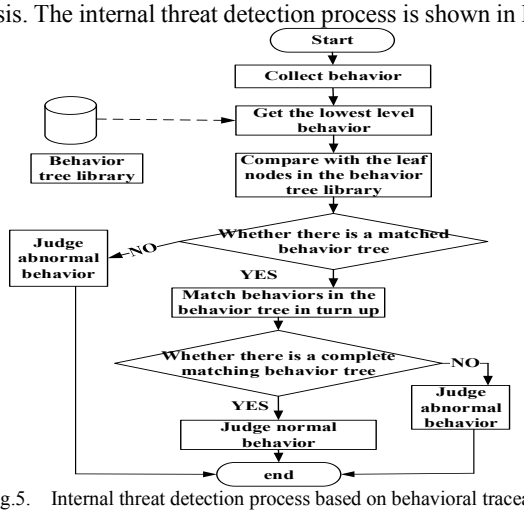

Fig.5. Internal threat detection process based on behavioral traceability

B. *Building normal behavior tree*

TABLE I. BEHAVIOR TREE BUILDING ALGORITHM FOR MULTI-LEVEL API ASSOCIATION ANALYSIS

Input: Specify virtual machine service behavior Useraction, keyword library and Openstack, part of source code in the libvirt

Output :Behavior tree built according to the specified behavior

Begin

1. Find *Useraction* from the behavior comparison library;
2. Establish the root node
   rootNode = userActionMap.getvalue(*Useraction*);
3. Go to the nova-api layer and read the api file;
4. Establish node by behavior action and nova-api library
   novaApiNode = novaApiMap.getvalue(*behaviorAction*)
   rootNode.addSon(*novaApiNode*);
5. Enter the manager file in the nova-compute layer;
6. Establish node by nova-api action and nova-compute library
   novaComputeNode=novaComputeMap.getvalue(*novaApiAction*)
   novaApiNode.addSon(*novaComputeNode*);
7. Enter the driver file in the libvirt-api layer;
8. Establish node by nova-compute action and libvirt-api library
   libvirtApiNode=libvirtApiMap.getvalue(*novaComputeAction*);
9. Enter the qemu-driver file in the libvirtaction layer;
10. Establish node by nova-compute action and qemu-driver library
    qemuDriverNode=qemuDriverMap.getvalue(*libvirtApiAction*)
    libvirtApiNode.addSon(*qemuDriverNode*);
11. Returns the full behavior tree of the current service *rootNode*;

End

We need to describe every normal behavior process of users calling Openstack cloud service using the behavior tree, where each layer of the behavior tree represents the operation of each layer in the cloud environment. First, we need to define the keywords of each layer of behavior-related source code, which leads to the keyword library corresponding to each layer of behavior. Secondly, after getting the user's operation behavior, we traverse the various levels of the cloud service, match the source code and behavior-related functions from the keyword library, and create corresponding behavior tree nodes through multi-layer API association analysis and logic. Finally, a leaf behavior node is created to complete the construction of normal behavior tree. The specific algorithm is shown in Table I.

Using the algorithm in Table I., a behavior tree instance for a relatively typical user operation such as creating a snapshot is established as shown in Fig.6.

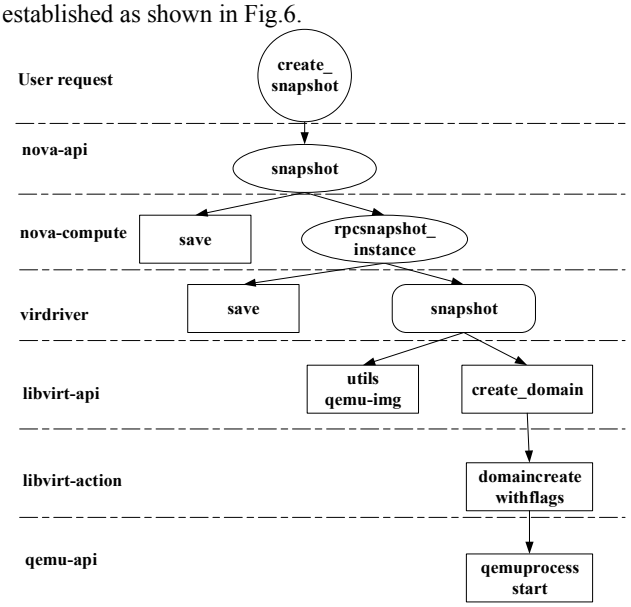

Fig.6. Normal behavior tree for users to create virtual machine snapshots

C. *Behavior acquisition*

Based on the analysis of the source code of the IaaS cloud platform, the computing service interface, management implementation interface, virtualization management interface, and virtualization process in Openstack are closely related to virtual machine services. In this section, we need to add a behavior collection point at the above interface. Specifically, we need to add a collection point at each behavior-related interface. For example, we can output the time of operation record when the specific calling behavior interface name on the computing service interface or the virtualization management interface（log.(time+{api}）). In addition, because the call to the virtualization process is relatively on the low-level, it is impossible to collect its behavior directly in user space by modifying the source code. Therefore, with the help of our previous work [40], the file access hook function is set at the bottom level to monitor the system calls related to the user's image file access.

The virtualization process monitoring algorithm is shown in Table II.

TABLE II THE CORE ALGORITHM OF THE MONITORING OF VIRTUALIZATION PROCESS

Input: The operation of accessing to the file by progress
Output: The information about the operation of virtualization process

Begin
1. Define a protected image file *protect_file*[][] = *new Files*[][];
2. Define the virtualization process *virt_current*[][]
   virt_current[][] = *new Process*[][];
3. Intercept file reading and writing operations to kernel hook functions
   readHook = *new Hook*()
   readHook.load()
   writeHook = *new Hook*()
   writeHook.load();
4. Get the process information of the currently reading and writing files current
   current = *readHook.getCurrent*()+*writeHook.getCurrent*();
5. Get reading and writing file information *file_path*
   file_path[] = *current.getFilePath*();
6. if(file_path in protect_file[][])
7. if (current in virt_current[][])
8. printk("Time : %d file:%s, current: %s\n", tm, filp->path, current)
9. return 0

End

By setting behavior collection points at the virtualization process in the Openstack cloud environment, we can obtain the behavior record of the user accessing the image file when the virtual machine related service is invoked in the user layer or the cloud environment intermediate node. The behavior information includes the information of the process of underlying virtualization, which can be obtained in the computing service log of the control node, the management implementation log of the computing node, the virtualization management log and the LSM hook function log.

After getting the behavior information, we need to get the behavior in each process. And we need to use the collection algorithm to create a complete behavioral process to be detected, and provide the necessary behavior information for the threat identification method. The algorithm for collecting behavior to be detected is shown in Table III. Different from creating a normal behavior tree, we need to use the "bottom-up" method to generate the information of behaviors that need to be detected based on behavior collection.

TABLE III ALGORITHM FOR COLLECTING BEHAVIOR THAT NEED TO BE DETECTED

Input: Various log files of operational behavior
Output: The information of behavior that need to be detected and related to the current operation

Begin
1. Find *libvirtApiAction* from the virtualization layer log file;
2. create a leaf node
   leafNode = *libvirtApi_log.get*(*libvirtApiAction*);
3. Find *lsmAction* in the log file of the LSM kernel of the virtualization management layer
   libNode = *lsm_log.get*(*lsmAction*);
4. If yes, established the virtualized management layer node
   if(*libNode*==*null*)
      return *leafNode*
   else
      leafNode.parent = *libNode*;
5. Find *managerAction* in the log file of the management implementation layer
   managerNode = *manager_log.get*(*managerAction*);
6. If yes, established the management implementation layer node
   if(*managerNode*==*null*)
      return *libNode*
   else
      libNode.parent = *managerNode*;
7. Find *calculationAction* in the log file of calculation service layer
   calculationNode = *calculation_log.get*(*calculationAction*);
8. If yes, established the calculation service layer node
   if(*calculationNode*==*null*)
      return *managerNode*
   else
      managerNode.parent = *calculationNode*;
9. Find *userAction* in the log file of the user layer
   userNode = *user_log.get*(*userAction*);
10. If yes, established the user layer node
   if(*userNode*==*null*)
      return *calculationNode*
   else
      calculationNode.parent = *userNode*;
11. Return the collected behavior *userNode*;

End

According to the above algorithm, we can get the information of behavior that need to be analyzed and related to the previous operation from the Openstack logging files.

D. Behavior Detection

In the Openstack cloud environment, we need to set the behavior tree matching flag and the level pointer. Then we need to get the underlying behavior of the current collection of behavior information, and then traverse all the behaviors of the behavior tree in the behavior library to find the matching behavior tree. If the current layer is completely matched, we need to continue to match upwards. If the behavior does not match completely, we need to modify the flag bit and jump out of the current logic and return the abnormal behavior flag and the level at which the exception call occurs; If it is exactly matched to a complete behavior tree at the end, it returns the flag of normal behavior and jumps out of the current program. The internal threat discovery algorithm based on behavior traceability is shown in Table IV.

TABLE IV THE DETECTION ALGORITHM OF BEHAVIOR TRACEABILITY

Input: Behavioral information that already hierarchically collected, normal behavior tree library, action and initiated time node
Output: Determine whether it is abnormal or not

Begin
1. Set the behavior tree matching flag
   flag = 1;

2. define a hierarchical pointer to the lowest level
   detectPointNode = userTree.getLeafNode
   normalNode = normalTree.getLeafNode;
3. Get the lowest level of behavior based on time
   userTime = detectPointNode.getTime
4. Get the behavior tree in the behavior library in turn, If there is a upper layer behavior node and time matching, iteratively get the parent node of the behavior
   while(*detectPointNode.parentNode*!=*null*){
   if(*userTime*==*normalNode.time*&&*detectPointNode.behavior*==*normalNode.behavior*){
   detectPointNode =*detectPointNode.getParent*
   normalNode = normalNode.getParent
   }
   else{
   flag = 0;
   currentLayer = normalNode.getCurrent;
   break;
   }
5. process match flag and return current info
   if (*flag*==1)
   return "*Normal behavior*"
   else
   return "*Abnormal behavior*"+current layer

End

V. EVALUATION

A. Experimental environment

We have deployed BTDetect on the Openstack cloud platform, the specific architecture is shown in Fig. 7. The version of Openstack is kilo. The Openstack cloud platform is divided into control nodes, computing nodes, network nodes and storage nodes. Related to virtual machine services are control nodes and computing nodes. The virtual machine service scenario is provided by using the Openstack simulation cloud platform and the normal behavior tree creation module obtains a normal behavior tree library by analyzing the Openstack source multi-layer API association; The behavior collection module collects behavioral actions of accessing user data at different levels; The behavior detection module is responsible for tracking and matching the collected behavior with the normal behavior tree to know whether the current behavior is abnormal. Therefore, through the trace analysis of the current user data access behavior, the discovery of the malicious behavior of the IaaS cloud environment is completed.

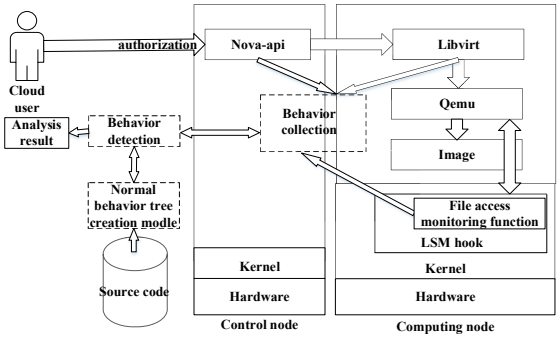

Fig.7. Implementation of BTDetect in Openstack

The Openstack version used in the experiment is the version of Kilo. The Openstack environment consists of two servers, which is a controller node and a compute node. It has the following configuration: 8GB Memory, Intel(R) Core i5-4590 CPU @ 3.30GHz processor and Centos 7 64-bit Operating system.

B. Effectiveness

In this section, we experimented with the effectiveness of the behavior acquisition module and the threat identification module. It is necessary to simulate a malicious internal person to make a call to the cloud service. In the experiment, we protected a user's image file with the image name 36c97be5-8fea-432a-99f8-la740926ee51.

We have made unauthorized remote calls to the operation of shutdown. Because the behavior tree level of the operation of shutdown involves the nova-api, libvirt, and qemu processes, we only need to collect behaviors that involves the computing service nova, the virtualization management libvirt, and the virtualization process qemu-kvm, as shown in Fig.8. It can be seen that there was authorized operation by the user 4da* in the time 2021-4-18 09:23, 09:45, 12:44, 12:47, 12:48 in nova log.

```
2021-4-18 09:23:54.632 1696 INFO nova.virt.libvirt.driver [req-8979aed4-349e-41a4-a485-lf48
f5444ce9 4da38c396569447a26845625c40f68 8209dc060fb94b447a905b74b5685b316 - - -] [instance:
36c97be5-8fea-432a-99f8-la740926ee51] Instance shutdown successfully after 3 seconds.
2021-4-18 09:45:37.632 1696 INFO nova.virt.libvirt.driver [req-8979aed4-349e-41a4-a485-lf48
f5444ce9 4da38c396569447a26845625c40f68 8209dc060fb94b447a905b74b5685b316 - - -] [instance:
36c97be5-8fea-432a-99f8-la740926ee51] Instance shutdown successfully after 3 seconds.
2021-4-18 12:44:44.632 1696 INFO nova.virt.libvirt.driver [req-8979aed4-349e-41a4-a485-lf48
f5444ce9 4da38c396569447a26845625c40f68 8209dc060fb94b447a905b74b5685b316 - - -] [instance:
36c97be5-8fea-432a-99f8-la740926ee51] Instance shutdown successfully after 3 seconds.
2021-4-18 12:47:11.632 1696 INFO nova.virt.libvirt.driver [req-8979aed4-349e-41a4-a485-lf48
f5444ce9 4da38c396569447a26845625c40f68 8209dc060fb94b447a905b74b5685b316 - - -] [instance:
36c97be5-8fea-432a-99f8-la740926ee51] Instance shutdown successfully after 3 seconds.
2021-4-18 12:48:49.632 1696 INFO nova.virt.libvirt.driver [req-8979aed4-349e-41a4-a485-lf48
f5444ce9 4da38c396569447a26845625c40f68 8209dc060fb94b447a905b74b5685b316 - - -] [instance:
36c97be5-8fea-432a-99f8-la740926ee51] Instance shutdown successfully after 3 seconds.
```

Fig.8. The results about the collection information of NOVA Node behavior

Next, the libvirt behavior information is analyzed. The result is shown in Fig.9. Since libvirt's time zone is the western time zone, log time plus 8 hours is Beijing time. That means, the libvirt behavior corresponds to the nova log at 09:23, 09:45, 12:44, 12:47, 12:48 on 2021-4-18. But at 12:35, 12:39, 12:40, 12:44, the libvirt behavior could not correspond to the nova log. This proves that the behavior records of these virtual machines are emitted directly through libvirt operations and not through the authorized nova component.

```
[root@compute qemu]# cat instance-0000002a.log | grep shutting'\ 'down
2021-4-18 01:23:53.974+0000: shutting down
2021-4-18 01:40:06.937+0000: shutting down
2021-4-18 01:41:32.769+0000: shutting down
2021-4-18 01:42:36.560+0000: shutting down
2021-4-18 01:45:36.828+0000: shutting down
2021-4-18 04:35:06.621+0000: shutting down
2021-4-18 04:39:33.573+0000: shutting down
2021-4-18 04:40:24.572+0000: shutting down
2021-4-18 04:44:43.481+0000: shutting down
2021-4-18 04:47:10.699+0000: shutting down
2021-4-18 04:48:48.943+0000: shutting down
```

Fig.9. The collection results of the information of the Libvirt node behavior

Finally, we need to analyze the behavior of the qemu virtualization process and use dmesg to view the output of kernel.

```
[ 6351.870859] Openning file name is disk !
[ 6351.870860] current:qemu-system-x86
[ 6351.870861] Time :2021-4-18 12:21:56
[ 6351.870862] Path:/var/lib/nova/instances/36c97be5-8fea-432a-99f8-1a740926ee51/disk,openn
ing file name is disk !
[root@compute nova]# cat nova-compute.log | grep 2021-4-18'\ '12:21
[root@compute nova]#
```

Fig.10. The collection results of the information of the Qemu node behavior

As shown in Fig.10, you can see a behavior record of a virtualization process at 2021-4-18 12:21. However, there is no corresponding behavior record when analyzing the behavior of the upper layer computing service interface. This proves that the current virtualization behavior is emitted by invoking the qemu process rather than the authorized Nova component.

The generated normal behavior tree of the user shutdown operation according to the previous multi-layer API association analysis is as shown in Fig.11. After the user's request is sent, it is first forwarded to the libvirt tool layer through the nova-api layer and then it is implemented by invoking the process of qemu virtualization. Then we need to use an internal threat discovery algorithm based on behavioral tracing. Then we input the action that needs to be detected and the time node that the action is sent.

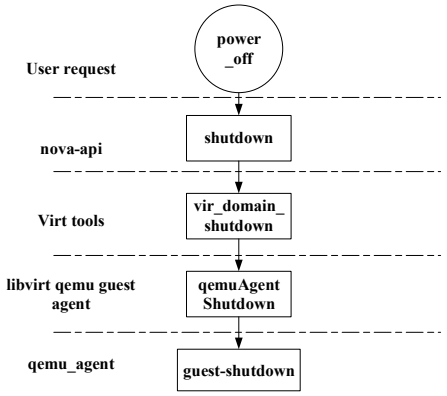

Fig.11. normal behavior tree about the shutdown of user virtual machine

According to the algorithm in Table IV, the result can be obtained as shown in Fig.12. The "shutdown" behavior at 09:45 and 12:44 is authorized, while the behavior at 12:35 and 12:21 is unauthorized. The reasons are as follows: The behavior tree constructed by the virtualization behavior at time 12:35 is not complete, so we consider it a malicious invocation behavior. We judged that malicious insider invoked virtualization management tools to pose a threat to user image files without authorization. Similarly, 12:21 is a malicious behavior of calling the virtualization process; At 09:45 and 12:44, the behavior information is consistent with the relevant behavior tree, so it is the normal behavior of the user. The experimental results show that behavior acquisition module and behavior detection module can be combined to detect the malicious behavior that internal users call cloud services to access user data.

```
[root@compute nova]# ./detector '2021-4-18 09:45' 'shutdown'
This access is authorized!
[root@comoute nova]# ./detector '2021-4-18 12:44' 'shutdown'
This access is authorized!
[root@compute nova]# ./detector '2021-4-18 12:35' 'shutdown'
This access is unauthorized!
The action starting point is libvirt level!
[root@compute nova]# ./detector '2021-4-18 12:21' 'shutdown'
This access is unauthorized!
The action starting point is qemu level!
[root@compute nova]#
```

Fig.12. The detection results of internal malicious threats

C. Accuracy

In this section we test the accuracy of the discovery of malicious internal threat. We chose some typical operations of users, "power on, power off, pause, lock, snapshot." We will call different levels of interfaces or processes such as computing service interfaces, computation management interfaces, virtualization management interfaces, virtualization processes, etc. to simulate malicious calls by malicious insiders at multiple levels. In the experiment, we performed 50 rounds of detection for each operation and count the detection accuracy of malicious behavior.

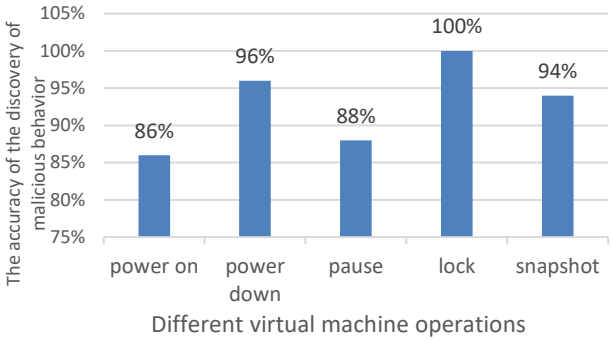

Fig.13. The accuracy of the discovery of malicious behavior for different virtualization operations

As shown in Fig.13, it can be seen that the malicious behavior recognition accuracy of the locking operation is 100%, but the operations such as power-on, power-off, pause, and snapshot do not reach the complete recognition accuracy. The reasons are as follows. In the behavior tree library established in Section IV.B, the locking operation does not overlap with other operating behavior tree nodes or exist as a subtree with other operation behaviors. That is, the locking operation is not a sub-operation of other operations, so the detection based on behavior traceability can obtain 100% recognition accuracy. However, operations such as power-on, power-off, pause, and snapshot may overlap with other operating behavior tree nodes or even subtrees. That is, the above virtualization operation may constitute a sub-operation of other operations or include other operations. Therefore, in the threat identification analysis of such operations, there will be conflicts with other operations. Although it is not completely accurate, the overall threat identification accuracy is about 90%.

Currently, the behavior tree is constructed semi-manually, and ongoing work will explore automated approaches based on machine learning or some heuristic classification algorithm such as [41].

VI. Conclusion

The transparency of cloud services makes users lose absolute control over private data, which make cloud environment security issues more challenging than traditional computing environments. One of the main security issues is the security of the virtualized environment. Although extensive research has been done on the security of cloud virtualization environments, little has been done to specifically focus on internal attacks in cloud environments. In this paper, we propose BTDetect, which can detect the malicious behavior of malicious internal personnel in the IaaS environment to illegally invoke the cloud service to access user data. We have implemented BTDetect in a real Openstack cloud environment and the experimental results show that BTDetect can effectively identify those threats that internal personnel maliciously invoke cloud services to access user data and has a high recognition rate.

REFERENCES

[1] Gai K, Guo J, Zhu L, et al. "Blockchain Meets Cloud Computing: A Survey". IEEE Communications Surveys & Tutorials, 2020, PP(99):1-1.

[2] K Costello, and M Rimol. Gartner Forecasts "Worldwide Public Cloud End-User Spending to Grow 18% in 2021," [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2020-11-17-gartner-forecasts-worldwide-public-cloud-end-user-spending-to-grow-18-percent-in-2021.

[3] "Top Threats to Cloud Computing: Egregious Eleven," Cloud Security Alliance. [Online]. Available: https://cloudsecurityalliance.org/press-releases/2019/08/09/csa-releases-new-research-top-threats-to-cloud-computing-egregious-eleven/.2021.

[4] "Network Security Events," Cloud Computing Security. [Online]. Available: http://cloud.idcquan.com/yaq/134034.shtml.

[5] M Ali, R Dhamotharan, E Khan, S U Khan, A V Vasilakos, K Li, and A Y Zomaya. "SeDaSC: Secure Data Sharing in Clouds," IEEE Systems Journal 11(2): 395-404. 2017.

[6] M Kazim, R Masood, and M A Shibli. "Securing the virtual machine images in cloud computing," In proceedings of the 6th International Conference on Security of Information and Networks. 2013.

[7] A Pandey, and S Srivastava. "An approach for virtual machine image security," In proceedings of International Conference on Signal Propagation and Computer Technology, pp. 616-623. IEEE. 2014, July.

[8] C.Tan, Y Xia, H Chen, and B Zang. "Tinychecker: Transparent protection of vms against hypervisor failures with nested virtualization," In proceedings of IEEE/IFIP 42nd International Conference on Dependable Systems and Networks Workshops (DSN-W), pp. 1-6. IEEE. 2012, June.

[9] Y Xia, Y Liu, and H Chen,. "Architecture support for guest-transparent VM protection from untrusted hypervisor and physical attacks," In proceedings of the 19th International Symposium on High Performance Computer Architecture, pp. 246-257. IEEE. 2013, February.

[10] T Miu,. "Research on Operating System Security and Performance in Virtualized Environments," (Unpublished master dissertation). Shanghai JiaoTong University, Shanghai, China. 2015

[11] S Paul, and S Mishra. "LAC: LSTM AUTOENCODER with Community for Insider Threat Detection," In proceedings of the 4th International Conference on Big Data Research, pp. 71-77. ACM. 2020, November.

[12] P Lou, Y Yang, and J Yan. "An Anomaly Detection Method for Cloud Service Platform," In proceedings of the 4th International Conference on Machine Learning Technologies, pp. 70-75. ACM. 2019, June.

[13] M Gupta, D. K Srivastava, D. S and Chauhan. "Security challenges of virtualization in cloud computing," In proceedings of the Second International Conference on Information and Communication Technology for Competitive Strategies, pp. 1-5. ACM. 2016, March.

[14] T Zhang, and R. B Lee. "CloudMonatt: An architecture for security health monitoring and attestation of virtual machines in cloud computing," Int'l Symposium on Computer Architecture.Vol.43, pp.362-374. ACM. 2015.

[15] Z Zhou, L Wu, Z Hong, M Xu, and F Pan. "Dtstm: dynamic tree style trust measurement model for cloud computing," Ksii Transactions on Internet & Information Systems, 8(1): 305-325. 2014

[16] C Cheh, U Thakore, A Fawaz, B Chen, W G Temple, and W H Sanders. "Data-driven model-based detection of malicious insiders via physical access logs," ACM Trans. on Mod. and Com. Simu. 29(4): 1-25. 2019.

[17] W Wang, X Du, N Wang, et al. "Review of Cloud Computing Security Audit Technology," Computer Science, 44(7): 16-20. 2017.

[18] Z Masetic, K Hajdarevic, and N Dogru. "Cloud computing threats classification model based on the detection feasibility of machine learning algorithms," IEEE 40th Int'l Convention on Information and Comm, Technology, Electronics and Microelectronics, pp. 1314-1318. 2017.

[19] D Bhamare, T Salman, M Samaka, A Erbad, and R Jain. "Feasibility of supervised machine learning for cloud security," IEEE Int'l Conf. on Information Science and Security (ICISS), pp. 1-5. 2016, December.

[20] M Aldairi, L Karimi, and J Joshi. "A trust aware unsupervised learning approach for insider threat detection," In proceedings of the 20th International Conference on Information Reuse and Integration for Data Science, pp. 89-98. IEEE. 2019, July.

[21] P Mishra, E S Pilli, V Varadharajan, and U Tupakula. "Securing virtual machines from anomalies using program-behavior analysis in cloud environment," IEEE HPCC/SmartCity/DSS Conf., pp. 991-998, 2016.

[22] H Tian, Z Chen, C C Chang, M Kuribayashi, Y Huang, Y Cai, Y Chen, and T Wang. "Enabling public auditability for operation behaviors in cloud storage," Soft Computing, 21(8): 2175-2187. 2017.

[23] Z. Shao, M. Wang, Y. Chen, et al., "Real-time dynamic voltage loop scheduling for multi-core embedded systems," IEEE Trans. on Circuits and Systems, 54 (5), 445-449, 2007

[24] M. Qiu, Z. Ming, J. Li, S. Liu, B. Wang, Z. Lu, "Three-phase time-aware energy minimization with DVFS and unrolling for chip multiprocessors," Journal of Systems Architecture 58 (10), 439-445, 2012

[25] M. Qiu, E. Khisamutdinov, et al., "RNA nanotechnology for computer design and in vivo computation," Philosophical Transactions of the Royal Society A, 2013

[26] K. Zhang, J. Kong, M. Qiu, G. Song, "Multimedia layout adaptation through grammatical specifications," Multimedia Systems 10 (3), 245-260, 2005

[27] L. Tao, S. Golikov, et al., "A reusable software component for integrated syntax and semantic validation for services computing," IEEE Symposium on Service-Oriented System Engineering, 127-132, 2015

[28] M. Qiu, Z. Ming, J. Li, J. Liu, G. Quan, Y. Zhu, "Informer homed routing fault tolerance mechanism for wireless sensor networks," J. of Systems Archi. 59 (4-5), 260-270, 2013

[29] J. Li, M. Qiu, J. Niu, et al., "Feedback dynamic algorithms for preemptable job scheduling in cloud systems," IEEE/WIC/ACM conf. on Web Intelligence, 2010

[30] Z. Zhang, J. Wu, et al., "Jamming ACK attack to wireless networks and a mitigation approach," IEEE GLOBECOM conf., 1-5, 2008

[31] X. Tang, K. Li, et al., "A hierarchical reliability-driven scheduling algorithm in grid systems," Journal of Parallel and Distributed Computing 72 (4), 525-535, 2012

[32] K. Gai, M. Qiu, X. Sun, H. Zhao, "Security and privacy issues: A survey on FinTech," International Conference on Smart Computing and Communication, 236-247, 2016

[33] K. Gai, M. Qiu, L. Chen, M. Liu, "Electronic health record error prevention approach using ontology in big data," IEEE HPCC conf., 2015

[34] H. Su, M. Qiu, H. Wang, "Secure wireless communication system for smart grid with rechargeable electric vehicles," IEEE Communications Magazine 50 (8), 62-68, 2012

[35] J. Niu, C. Liu, et al., "Energy efficient task assignment with guaranteed probability satisfying timing constraints for embedded systems," IEEE Trans. on Parallel and Distributed Systems, 25 (8), 2043-2052, 2013

[36] Y. Guo, Q. Zhuge, J. Hu, et al., "Data placement and duplication for embedded multicore systems with scratch pad memory," IEEE Trans. on CAD, 2013

[37] H. Zhao, M. Chen, et al., "A novel pre-cache schema for high performance Android system," Future Generation Computer Systems 56, 766-772, 2016

[38] K. Gai, M. Qiu, B. Thuraisingham, L. Tao, "Proactive attribute-based secure data schema for mobile cloud in financial industry," IEEE 17th HPCC, 2015

[39] K. Gai, M. Qiu, H. Zhao, J. Xiong, "Privacy-aware adaptive data encryption strategy of big data in cloud computing," IEEE 3rd CSCloud conf., 2016

[40] L. Lin, S. Li, B. Li, J. Zhan, Y. Zhao. "TVGuarder: A trace-enable virtualization protection framework against insider threats for IaaS environments," In Cyber Security and Threats: Concepts, Methodologies, Tools, and Applications, pp. 638-658. IGI Global. 2018.

[41] Y. Hua, K. Gai, Z. Wang, "A Classification Algorithm Based on Ensemble Feature Selections for Imbalanced-Class Dataset," In IEEE BigDataSecurity conf.. 2016.