Int J Syst Assur Eng Manag (June 2018) 9(3):567–576
https://doi.org/10.1007/s13198-014-0277-7

ORIGINAL ARTICLE

HIDS: A host based intrusion detection system for cloud computing environment

Prachi Deshpande · S. C. Sharma · S. K. Peddoju · S. Junaid

Received: 2 May 2014 / Revised: 3 June 2014 / Published online: 24 June 2014

Abstract The paper reports a host based intrusion detection model for Cloud computing environment along with its implementation and analysis. This model alerts the Cloud user against the malicious activities within the system by analyzing the system call traces. The method analyses only selective system call traces, the failed system call trace, rather than all. An early detection of intrusions with reduced computational burden can be possible with this feature. The reported model provides security as a service (SecaaS) in the infrastructure layer of the Cloud environment. Implementation result shows 96 % average intrusion detection sensitivity.

Keywords Cloud · Detection · Host based IDS · OpenNebula · Network · Security · System call traces · Virtual machine

P. Deshpande (✉) · S. C. Sharma
Department of Applied Science & Engineering, Indian Institute of Technology Roorkee, Roorkee, Uttarakhand 247667, India
e-mail: deprachi3@gmail.com

S. C. Sharma
e-mail: scs60fpt@iitr.ac.in

S. K. Peddoju · S. Junaid
Department of Computer Science & Engineering, Indian Institute of Technology Roorkee, Roorkee, Uttarakhand 247667, India
e-mail: drpskfec@iitr.ac.in

S. Junaid
e-mail: siddiqui.mohdjunaid@gmail.com

1 Introduction

The advent of Cloud is a milestone in technological advancement for speedy information processing. With the introduction of a new computing system, its security issue becomes a prime concern for academicians and researchers. To secure the information processing across any information system has become pivotal in the success of an information processing system.

Cloud computing provides a rapid and location independent information processing. Due to location independent processing, trust is one of the major issues among the Cloud users for using its resources. Hence Cloud security becomes essential for successful deployment of its services. Due to security apprehension, a third party security service is not attractive in comparison of an in built security mechanism. This is the area where the intrusion detection system (IDS) fits in. The ideal IDS is the one which has 100 % detection efficiency against the possible vulnerabilities. It can be designed based on detection techniques, deployment location, and alert mechanism (Abraham et al. 2007; Modi et al. 2013). The intrusions can be detected by anomaly or signature based detection techniques. The signature detection based IDS cannot identify the novel attacks as it is based on known signatures. The anomaly based detection technique uses the deviation in the established pattern of a particular user to identify the intrusion. The only drawback with this technique is the high false-positive rate of detection and can be overcome by suitable classification method. Based on location of deployment, IDS can be host-based or network based entity. Host stationed IDS (HIDS) completely depends on the target system itself, whereas network based IDS (NIDS) depends on the network environment.

An intruder can acquire the status of administrator (in Windows operating system (OS)) or root (UNIX/Ubuntu/Linux OS) by gaining the access of the privileged programs (Mukkamala et al. 2004). This flaw has been mitigated with the help of program profile generation by capturing the system calls. Hence, it will become difficult for an attacker to perform its activities without evading the execution logs. As a result, a program based profile creation is more stable as compared to the behavior based profile of a user for identification of an intrusion.

1.1 Present scenario

The notion of intrusion detection was first introduced by Anderson in 1980 (Anderson 1980) followed by the study of the first intrusion detection system model in 1987 (Denning 1987). Since then, with the advent in communication networks and methodologies, the secure data processing has become need of the hour. As far as IDS is concerned, the classification of various attacks is very crucial. Based on the classification, the IDS can generate the alerts to the user or the administrator against the unauthorised access or malicious activities.

There are various classifiers reported in the literature such as rule learning (Lee et al. 1997), Hidden Markov model (HMM) (Warrender et al. 1999). But HMM approach increases resource consumption. Further, k-nearest neighbor (kNN) (Payne et al. 1997), artificial neural networks (Ghosh et al. 1999), and a binary weighted cosine metric (Rawat et al. 2006) had been also reported in the literature as the classifiers. Till date, a little work has been reported for HIDS in Cloud environment. Forrest et al. (1996) had proposed a HIDS using a feed forward artificial network for analysis of behavior of users. But it was not verified against wireless environment. The authors had carried out the experimentations on the synthetic data sets. Using systems 'default log', self similarity measures were calculated by Wespi et al. (2000) for intrusion detection. But the effort was limited only to Windows OS.A standard '1998 DARPA BSM' data set had been used for this analysis. In another approach, IDS models were invoked according to the severity of the attacks (Tandon and Chan 2005).The prediction of intrusion had been carried out by the behavioral analysis of the user. This approach suffered from increased resource consumption according to the user's privilege. Neural network based anomaly detection (Ying et al. 2010) method's accuracy was based on creation of log files and also it was not verified over the Cloud environment. Statistical method based HIDS (Vokorokos and Balaz 2010) had been utilized for data evaluation wherein detection was based on the information of user activity deviation. HIDS for ARP based attack detection (Barbhuiya et al. 2011) was limited to a local area network environment. A data normalization approach (Cai et al. 2010) for anomaly detection had been also used in IDS. Agent based IDS had been proposed (Doelitzscher et al. 2012) for Cloud environment. Random forest (RDF) method (Htun and Khaing 2013) had been employed for prediction of anomalies along with an analysis using the standard 'KDD'99 dataset (KDD 1999).

It has been evidenced from the available literature that, till date, no effort has been initiated to verify the performance of the IDS in the real time environment. The research gaps, based on the reported methods, towards the deployment of HIDS can be summarized as—

(a) Most of the existing system had used artificial data sets, rather than the real-time data, for analysis purpose.
(b) The reported mechanism(s) had a very large training time for detection of the malicious activity.
(c) For identification of the intrusion, the existing mechanisms rely on all the system calls rather to be specific. Alert generation only after analyzing entire system call trace results into a slow or late response against the intrusion
(d) Real time IDS for Cloud with early detection of intrusion never considered in any of the reported methods.

1.2 Architecture of Cloud with HIDS

The research gaps available from the state-of-the-art steers to a conclusion that there is a need for a new method to determine the intrusion in real-time environment. Hence, in the present work, a HIDS has been initiated with real-time data analysis. Only a failed system call traces were used to predict the intrusion. This feature will reduce the burden in the IDS and generate the early prediction of the intrusion. The abnormal behavior has been predicted using kNN classifier. kNN is best suited for a distributed environment like Cloud computing due to its highly scalable nature.

The proposed IDS framework is based on traditional IDS with an improvement of adopting a modular approach and real-time analysis so as to make it work with Cloud infrastructure. Each component has been designed in a layered manner with a specific task to carry out. Figure 1 shows the architecture of proposed HIDS. It has a front-end machine with OpenNebula installation and a host machine that provides resources to virtual machine (VM). The user sends request to front-end for accessing the virtualized resources. Front-end creates VM for the user on the host. HIDS monitors the VM behavior using the available modules in the HIDS.

It is very difficult to supervise and identify intrusive events in the Cloud environment due to thousands of virtual and actual machines and allied inward traffic. Hence,

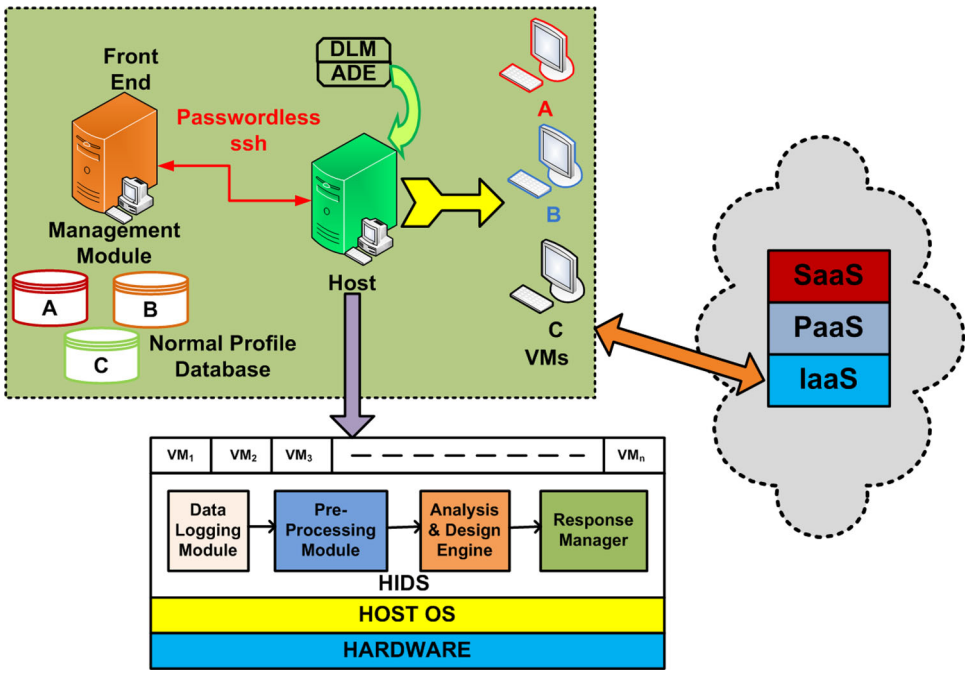

Fig. 1 Architecture of HIDS in Cloud Environment

each VM must be equipped with an IDS to enforce defense against internal and external vulnerabilities.

The anomaly detection requires audit logs that are generated on the target machine itself for identifying the intrusion. Hence, system call traces have been used for the purpose of audit logs to monitor the running processes on the system. It is provided by the OS running over that machine. Also, they are vulnerable to modification by the attacker. Hence it forces IDS to identify the attack before it could manipulate its activity traces as normal. In this work, for root (administrator) all the audit logs has been analyzed, whereas for a user, only audit logs of failed processes are analyzed. The motivation behind this strategy is, users unprivileged activities will be failed, which may be an intrusion. This act will minimize the response time for alert generation being selective in processes. Some of the host based information sources in 'Linux/Ubuntu OS' are as follows:

- ***Accounting***: It keeps the record regarding the resource usage, such as memory, disk, CPU, network usage and the application or processes invoked by the users present on the system
- ***Syslog***: It is an audit service made available by the OS to the application program to store the logs generated by them. It stores this log information along with the time stamp and process id of the corresponding application. Being a daemon process, it is always running in the system waiting for the information to be logged
- ***Linux audit***: Linux audit framework is shipped along with 'SUSE' enterprise Linux and Ubuntu. Audit enables users to perform various tasks such as mapping processes to user, generation of audit report using *'aureport'* tool, filtering of event of interest at different levels (user, process, group, system call etc.) and prevention of audit data loss

Traditional IDS has limitations to identify the intrusion due to unavailability of unknown attack signatures. Even if anomalies were detected, it lags in correct identification of them as intrusions. Hence, a mechanism is required, which not only identify the intrusions but also alerts the user very quickly against it. The distinguishing features of the proposed work from its counterpart's are:

1. Creation of an indigenous database of normal activities instead of standard data sets used in (Warrender et al. 1999; Rawat et al. 2006).
2. After process execution, entire trace of a process is not captured as the process terminated may be invasive one. Hence, a novel time interval based logging technique has been proposed to overcome this problem. This approach reduces the intrusion by identifying it at a very early stage. A kNN method has been used for comparing the current information with the available database.

The main steps in devising a simple framework for deploying HIDS over the Cloud can be summarized as:

Capturing and preprocessing: A module to capture the system calls trace of running process, filtering of raw data into useful information and store them in the database. Same module can be used to capture current system call traces.

Analysis: A module to match and analyze the information obtained after capturing and preprocessing to

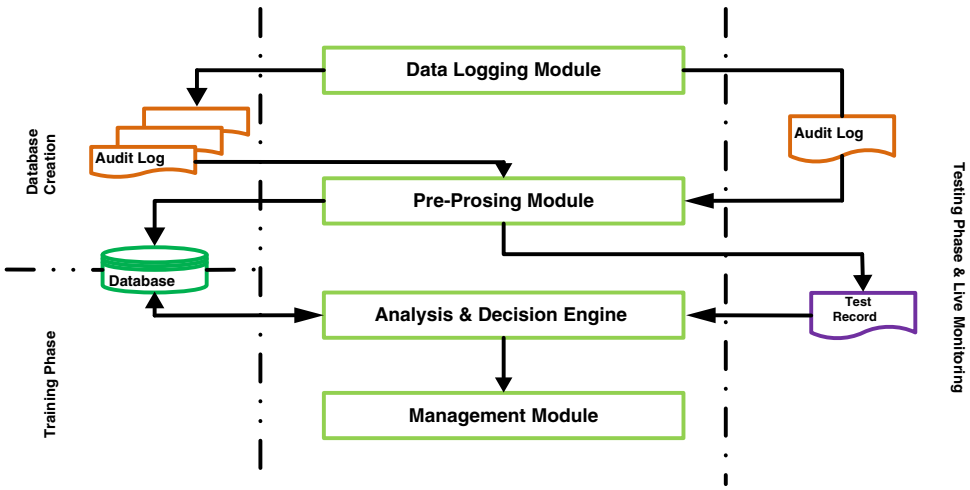

Fig. 2 Proposed component based model for IDS

identify anomalous behavior. Data mining techniques has been applied to perform this task.

Control and management: A monitoring unit to initiate suitable action according to the severity against anomalous behavior detected by the analysis component. Coordination with other IDS in the Cloud environment has been taken care by this unit.

The intrusion can be identified by the 'audit log'. Every system call has been recognized as a word and every execution of the program is treated as a document. With the help of kNN classifier, malicious activities can be identified. The rest of the paper has been organised as: The proposed work and its methodology are discussed in Sect. 2. The experimentation with their outcomes is reported in Sect. 3. The article concludes in Sect. 4 along with its future scope.

2 Modules of the proposed framework

The framework for integrating IDS with the OpenNebula private Cloud (Deshpande et al. 2013), intermediate steps in proposed IDS model development and deployment, and basic work flow of the complete system has been discussed in this section.

2.1 Proposed intrusion detection model

Security as a service (SaaS) in Cloud had already been investigated by many researchers. But an IDS as a service in a Cloud is hardly examined. Also, there is no such standard framework or architecture developed for setting up IDS in Cloud. Hence this attempt will help to the Cloud owner to provide IDS as a service. Figure 2 shows the component based model for the proposed IDS. The complete system has been divided into four modules.

2.1.1 Data logging module (DLM)

As the name suggests DLM is responsible for recording the audit logs generated by the application program and process running in the system. A huge information has been generated by the application programs for debugging purpose. But only useful information is recorded using filters and rules available in data logging components. System call trace can be carried out by two ways. A kernel module can be integrated with the kernel to intercept the system calls invoked by a user process. It reduces tracing overhead, but very complex to build. A simpler method is to use an accounting facility which is provided along with almost every Linux/Ubuntu distribution.

The second option has been chosen for the present work. For this purpose '*Linux audit*' framework is being used. It is an accounting utility shipped with 'SUSE' enterprise Linux distribution. This can also be installed in Ubuntu. Figure 3 depicts the Linux *'audit'* framework.

The various components of audit framework are summarized as:

- ***auditd***: This is an audit daemon continuously running in the background of the system. As soon as the system get started *'auditd'* starts writing the audit information to 'audit.log' generated by the kernel audit interface, processes and application activities. The initial configuration of the *'audited'* can be managed through its configuration file available in *'/*etc*./sysconfig/auditd'*. Once the *'auditd'* gets started it can be further controlled through *'/*etc*./auditd.conf'*
- ***audit rules***: This rule file is the core component for the proposed work. By placing appropriate rules, one can restrict the logging of only those system calls which are of interest to intrusion detection purpose. This rule file is loaded with the initiation of the audit daemon.

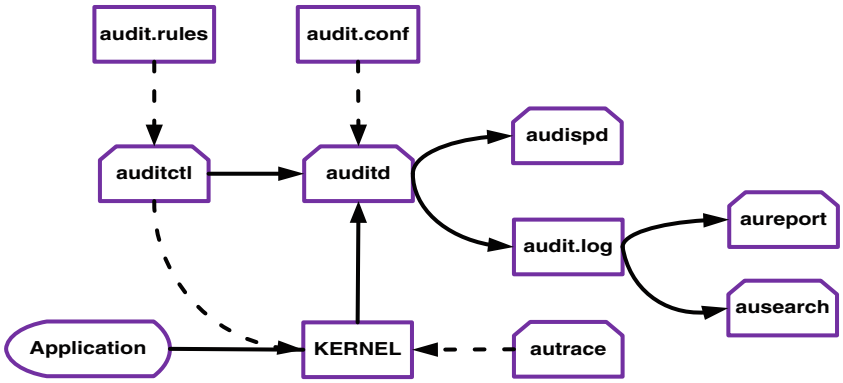

Fig. 3 Linux *'audit'* framework

- ***aureport***: This utility enables the administrator to generate custom report and extract useful information from the raw data logged in the log files. The output of *'aureport'* can be used in different application for visualizing the audit logs
- ***ausearch***: It allows the user to customize its search based on different filters like process id, user id, group id, system call name and various other keywords of the logged format

Using *'auditd'* framework, the failed system calls for a process are recorded with their frequency. This recording is carried out, at a time interval of 30 and 60 s, during process execution and can be expanded as per the requirements. Due to this novel time based logging technique, a process can be identified as normal or intrusive. This will enable early detection of intrusion.

2.1.2 Preprocessing module

To analyze each field in the log file, large disk space is required to store such logs. Also it is time consuming and makes resource exhaustion. Therefore, preprocessing has been initiated to filter out the important feature for populating the database which can be used later for analysis purpose. Figure 4 shows the workflow of the preprocessing module. The preprocessing has been carried out in three phases as:

- *Phase I*
  The logs obtained from the data logging module contains fields like record number, date, time, process id, system call name, process name, user name, etc. Out of all these information, only two column values i.e. process id (P_ID) and system call (syscal) are of most interest. Thus output of this phase is a table containing two columns with information of a specific process and its system call. This file is then processed in phase II of the preprocessing.
- *Phase II*
  In the second phase, the records are aggregated to calculate the frequency of system calls invoked by individual process. Here the frequency of each system call by each process is calculated as a process issues same system call many a times in span of its execution. The output of this phase contains a table containing three columns, namely, process id, system call, frequency. This information is then passed to the phase III.
- *Phase III*
  In the final phase, all the records has been converted into vectors representing each process with frequency distribution of system calls, ordered in a predefined format. Only a specific set of system calls (i.e. failed) are collected. Therefore, with respect to each process, a vector is obtained whose cell will contain the frequency for that particular system call.

2.1.3 Analysis and decision engine (ADE)

This is the core component of the proposed system. It verifies the test records against a database containing the reference records by applying data mining algorithm. Analyzing the system calls for intrusion can be mapped to the text categorization technique in which similarity between the documents has been calculated by measuring the extent of similarity between the words used in those documents. Various classification and machine learning techniques were used for text categorization such as regression models, Bayesian classifiers, decision trees, nearest neighbor classifiers, neural networks, and support vector machines (Aggarwal and Zhai 2012).

In text classification, the document with character strings has been converted into a form appropriate for the categorization work. A vector space model, in general, is used for representing the documents in which documents are transformed into vectors indicating the occurrence of words in those documents.

A matrix $\boldsymbol{X}$ is used for compilation of documents and given by $\boldsymbol{X} = (x_{ij})$. Here x_{ij} is the value of word i in document j. Boolean weighting is the most simple approach

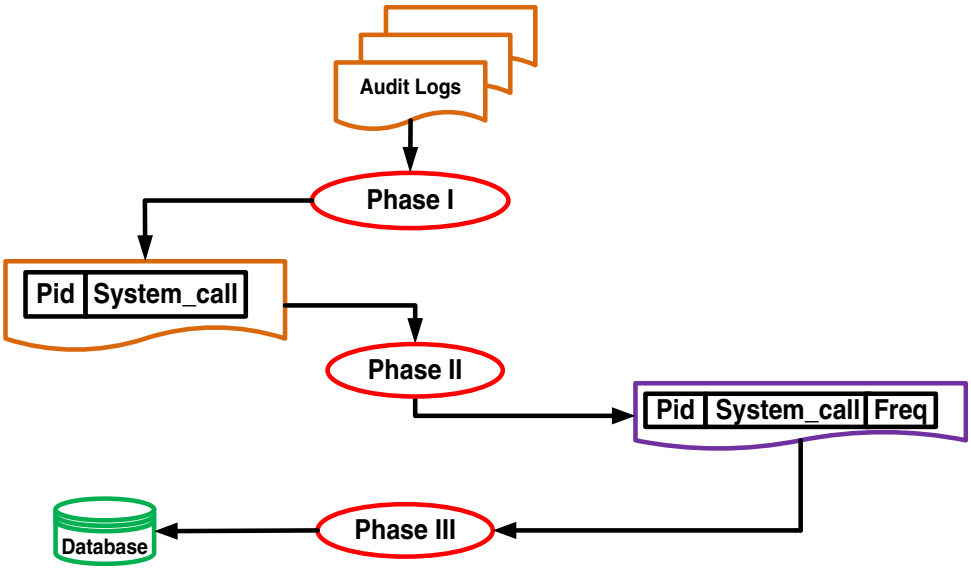

Fig. 4 Workflow of the preprocessing module

which sets the weight x_{ij} to 1 if the word is present in the text and otherwise 0.

In the present work, a kNN classifier has been employed. It works on the postulate that the categorization of nearby instances is analogous in a vector space. Compared to Bayesian classifier, kNN doesn't require prior probabilities as the Bayesian classifier does and hence is fast in terms of calculations. It is very easy to initiate recurrent additions in the training document and introducing new training documents with kNN classifier. This important aspect of kNN makes it suitable for a very dynamic and distributed environment of Cloud computing.

kNN classifier grades the neighbor vectors among the training document, and uses its labels of k most analogous neighbors to forecast the class of the new document. The similarity has been estimated with the help of Euclidean distance or the cosine value between two document vectors. The cosine similarity is defined as—

$$sim(X, P_j) = \frac{\sum_{t_i} \in (X \cap P_j) x_i \times p_{ij}}{\|X\|_2 \times \|P_j\|_2} \quad (1)$$

where X is the test document; Pj is the jth training document; t_i is a word shared by X and P_j; x_i is the weight of word t_i in X; p_{ij} is the weight of word t_i in document P_j; $\|X\|_2$ is the norm of X, and $\|P_j\|_2$ is the norm of P_j. A cutoff threshold is required to assign the new document to a known class.

These vectors are then stored in database which is a two dimensional matrix where each row represent a document and each column represents a word from the vocabulary. The value in a cell [i, j] represent the frequency of 'j'th word in 'i'th document. Intrusion detection using the system call trace of processes best fits for this kind of categorization. Hence technique used in this work is too based on this terminology. An analogy between intrusion detection using system call trace of processes and text categorization has been described in Tables 1 and 2.

Table 1 Document to word matrix

Doc_ID/word	Intrusion	Detection	System	…..	Cloud
1	0	1	2	….	1
2	1	1	1	….	3
….	….	….	….	….	….
3	2	3	1	….	0

Table 2 Process system call matrix

P_ID/syscall	Read()	Write()	Open()	…..	Exit()
1890	0	147	237	….	876
2089	152	145	178	….	533
….	….	….	….	….	…
3540	245	3	61	….	450

The vectors so obtained after preprocessing phase are analogous to vectors for documents where each process maps to a document and its information vector, containing the frequency of each system call for that process. The flow chart for analysis and detection of *auditlogs* has been given in Fig. 5.

2.1.4 *Management module*

The component '2.1.1 to 2.1.4' will be collectively deployed on the VM, whereas management module (MM) works at front end OpenNebula Cloud infrastructure (Deshpande et al.

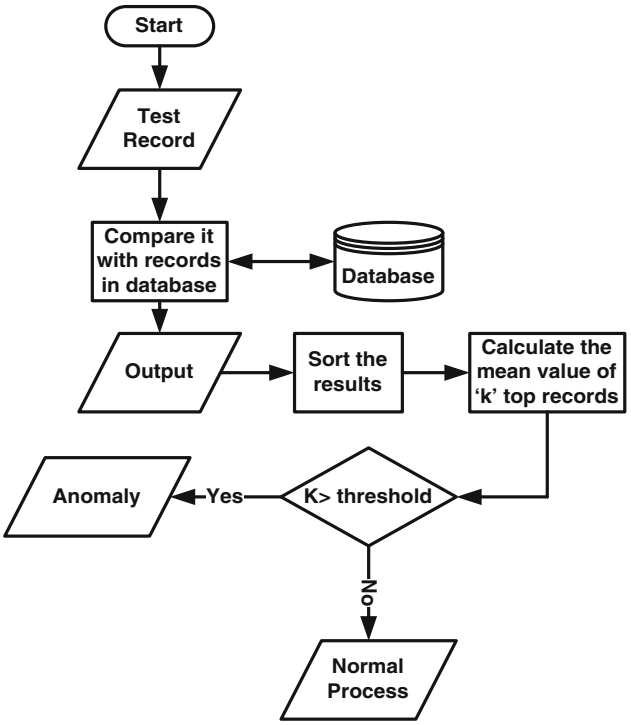

Fig. 5 Flow chart for analysis and detection of *'auditlogs'*

2013). The basic role of management module is to upload the normal profile database of user to its assigned VM at the time of system startup. Depending on the severity of intrusion attack, it will alert the VM user, or even shut it down. In case of any intrusion, the IDS running on VM reports to management module to take preventive actions which can vary from alerting the VM user, suspending a particular VM and even shutting down of VM.

Each VM on the Cloud will be shipped with a complete IDS system with mechanism to communicate with the management module present on front end of the OpenNebula private Cloud. The model has been designed to incorporate database creation, testing phase, training phase as well as live monitoring environment. Data logging module and preprocessing component will be in action for almost every phase.

Analysis and decision module will not be the part of database creation phase. Training phase is carried out using analysis and decision making module as well as the alert generation module. Testing phase is nothing but off-line working of IDS in which known data is pushed to evaluate the accuracy of the system and hence would cover all the components.

2.2 The work flow

The entire system starts with the creation of normal profile database for the user whose activities are to be monitored. This database creation is a one time process. It is carried out as soon as a new user is added i.e. a new request for Cloud resource arrives. All the activities were captured over less than a week time so as to define a normal behavior of the user. Once the database creation is done, it is stored at a repository in front end.

Then intrusion detection model undergoes training phase and testing phase before getting available for live deployment. In training phase, the database is tuned to the normal profile of the user. To evaluate the accuracy of the analysis and decision engine, testing is performed. In this phase audit logs of known processes are rated as normal or invasive. Analysis is performed over these records against the obtained database to check whether it is able to correctly identify the process as normal or intrusive. The algorithm of the proposed method is given in the Fig. 6.

3 Implementation and results

The experimentation and results has been discussed in this section. The system calls which were included in the dataset and used in the experiment has been listed out here.

3.1 Dataset—system calls

Systems calls are often seen as an interface between user space and kernel space. This distinction of space is maintained for security reasons. User space program can use the kernel services through the use of system calls. Thus system calls are the only way to break the barrier between these two spaces. The system calls are functions specific to kernel, they cannot be used directly in the user space program. Instead, APIs are provided to programmer through which the system call can be invoked. In order to change the mode from user to kernel execution, a software generated interrupt is used which is known as an "operating system trap". This interrupt is invoked by the inbuilt library functions provided by the compiler. The system calls are divided into different categories based on their functionality, like file system management, process management, intercrosses communication. The list of system calls which has been used for monitoring in this work is given in Table 3.

The results has been estimated using three different real-time datasets, with a time window of 30 and 60 s. For analysis of the available traces, a confusion matrix is created as given in Table 4. A higher value of 'True Positive' detection is desirable for robust IDS.

Further, the performance of the IDS is analyzed by using various cost functions such as accuracy, true positive rate, true negative rate, positive prediction value, negative prediction value, false positive rate, false negative rate, false discovery rate, F1score, informedness and markedness

Fig. 6 Algorithm for the proposed method

```
1. Construct the standard data set
2. for every process vector 'Xi' in test data set do
3. for each process vector 'Pj' in database set
4. Calculate distance (Xi,Pj);
5. If distance (Xi,Pj) equals to zero
6. 'Xi' is normal;
7. exit;
8. Sort the distance in increasing order;
9. Find 'k' top records and calculate their avg (dist);
10. If avg((dist)<threshold)
11. Xi is normal;
12. else
13. 'Xi' is abnormal;
```

Table 3 Summary of system calls used for analysis

Categories	Description	System calls name
File management	Create a channel	Creat()
	Open a file	Open()
	Close a file	Close()
	Read into a file	Read()
	Write into a file	Write()
	Random access	Lseek()
	Channel duplication	Dup()
	Aliasing a file	Link()
	Removing a link	Unlink()
	Status of a file	Stat(), fstat()
	Access control	Access(), chmod(), chown(), umask()
	Device control	Ioctl()
Process management	Process creation and termination	Exec(), fork(), vfork(), wait(), exit()
	Process ownership and group	Getuid(), geteuid(), getgid(), getegid()
	Process identification	Getpid(), getppid()
	Process control	Kill(), alarm()
	Change working directory	Chdir()
Inter process communication	Pipelines	Pipe()
	Messages	Msgget(), msgsnd(), msgrcv()
	Semaphores	Semget(), semop()
	Shared memory	Shmget(), shmat(), shmdt(), mmap(), munmap()

Table 4 Confusion matrix

Test outcome	Condition positive	Condition negative
Positive	True positive (TP)	False positive (FP)
Negative	False negative (FN)	True negative (TN)

(Fawcett 2006).Here threshold value(T_T) of 1, 10 and 20 are considered for the classification of the process as normal or intrusive. The system call traces are analyzed for a time frame of 30 and 60 s. The system call sequence for each new process can be scanned and extracted for every new process. After transformation into a vector, with the help of Eq. 1, resemblance between the new process and the normal data set can be calculated. For a similarity score 1, each new process is rated as normal. Otherwise, kNN is chosen to determine the status of a particular new process. Here the threshold of classification is set by considering the average similarity values of kNNs with highest similarity index. Any new process is considered as normal only when the average similarity value is above the threshold. During the verification, the proposed IDS compare each new process against the available data set. By estimating the Euclidian distance between the kNNs and the threshold value, the particular process has been classified as

Table 5 Confusion matrix for system call trace

Duration	T_T	Test	Condition positive	Condition negative	Total
30 s	1	Positive outcome	979	499	1,478
		Negative outcome	101	171	272
		Total			1,750
	10	Positive outcome	1,263	310	1,573
		Negative outcome	77	149	226
		Total			1,799
	20	Positive outcome	1,510	144	1,654
		Negative outcome	46	84	130
		Total			1,784
60 s	1	Positive outcome	1,180	496	1,676
		Negative outcome	87	201	288
		Total			1,964
	10	Positive outcome	1,429	241	1,670
		Negative outcome	106	212	318
		Total			1,988
	20	Positive outcome	1,711	103	1,814
		Negative Outcome	86	96	182
		Total			1,996

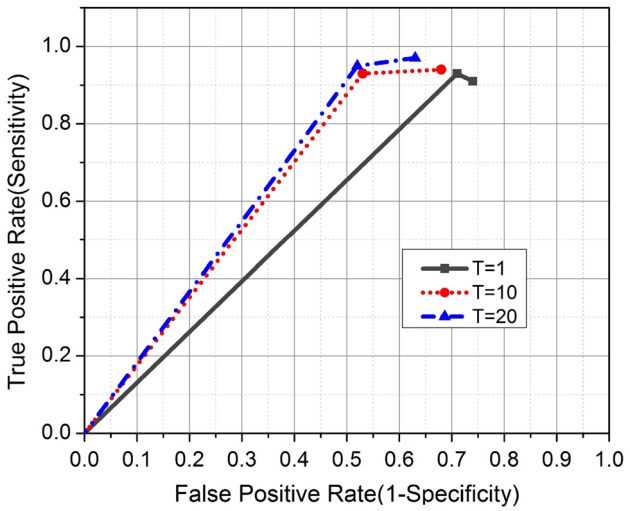

Fig. 7 ROC for the proposed IDS framework

Table 6 Result comparison

Parameter	Dataset-30 s			Dataset-60 s		
Threshold T_T	1	10	20	1	10	20
Sensitivity	0.91	0.94	0.97	0.93	0.93	0.95
Specificity	0.26	0.32	0.37	0.29	0.47	0.48
Precision	0.66	0.80	0.91	0.70	0.86	0.94
Negative predictive value	0.63	0.66	0.65	0.70	0.67	0.53
False positive rate	0.74	0.68	0.63	0.71	0.53	0.52
False discovery rate	0.34	0.20	0.09	0.30	0.14	0.06
Miss rate	0.09	0.06	0.03	0.07	0.07	0.05
Accuracy	0.66	0.78	0.89	0.7	0.83	0.91
F1 score	0.77	0.87	0.94	0.80	0.89	0.95
Informedness	0.16	0.27	0.34	0.22	0.40	0.43
Markedness	0.29	0.46	0.56	0.40	0.52	0.47

normal or else. The characteristics of the proposed method are summarized in Table 5.

Further the performance of the system had been analyzed with the help of receiver operating characteristics (ROC) and area under the curve (AUC) by using different threshold value. Figure 7 shows the ROC for the proposed system. The sensitivity of the proposed model is directly proportional to the threshold value. For a threshold value of 20, the proposed system shows a fair amount of accuracy as well as sensitivity. The performance of the system can be improved by a rigorous and continuous observation in the Cloud environment for updating the data logs in real-time.

From Table 6, it can be evidenced that the threshold value 'T_T' and the accuracy and true positive values are directly proportional to each other. The analysis of various cost function shows the profoundness of the proposed method. The average performance of the proposed system has been summarized in Table 7.

Table 7 The average characteristics of the proposed method

Dataset used	Threshold	Accuracy (%)	Sensitivity (%)	Specificity (%)	Maintenance
Real-time data capture	1	66	92	27.5	Overhead reduced due to deletion of VM logs when VM is deleted.
	10	80.50	93.5	39.5	
	20	90	96	42.5	

4 Conclusions

A HIDS, based on anomaly detection, for Cloud environment had been reported in this paper. Based on the assumption that anomalous behavior is evidently different from the normal behavior, normal profile for a Cloud user had been created using the system call trace of applications and programs running in the system. kNN classifier was used to classify the system call traces as it allows easy incorporation of new training document. This feature is very helpful in highly scalable Cloud environment. Also instead of monitoring successful system calls, frequency of failed system calls has been preferred for analysis. Detection accuracy with a high sensitivity of 96 % indicate a fair performance of the proposed method. With this method, accuracy can be increased but at the cost of delayed detection. In future, the present work can be extended to frame an adaptive management module for initiating preventive actions after intrusion detection and the integration of HIDS and NIDS with the help of updated data logs.

References

Abraham A, Grosan C, Martin-Vide C (2007) Evolutionary design of intrusion detection programs. Int J Netw Secur 4(3):328–339

Aggarwal C, Zhai C (2012) A survey of text classification algorithms. In: Mining Text Data. New york, Springer 163–222

Anderson J (1980) Computer security threat monitoring and surveillance, Technical report. James P. Anderson Co., Fort Washington

Barbhuiya F et al (2011) An active host-based intrusion detection system for ARP-related attacks and its verification. Int J Net Sec App 3(3):163–180

Cai L, Chen J, Ke Y, Chen T, Li Z (2010) A new data normalization method for unsupervised anomaly intrusion detection. J Zhejiang Uni-SCI C 11(10):778–784

Denning D (1987) An intrusion detection model. IEEE Trans Soft Eng 13(2):222–232

Deshpande P, Sharma S, Kumar S (2013) Implementation of a private cloud: a case study. Adv Int Sys Comp 259(2):635–648

Doelitzscher F et al (2012) An agent based business aware incident detection system for cloud environments. J Cloud Comp Adv Sys App 1–9. doi:10.1186/2192-113X-1-9

Fawcett T (2006) An introduction to ROC analysis. Patt Recog Lett 27:861–874

Forrest S, Hofmeyr A, Somayaji A, Longsta T (1996) A sense of self for Unix processes. IEEE Symp Security and Privacy, Oakland, pp 120–128

Ghosh A, Schwartzbard A, Shatz A (1999) Learning program behavior profiles for intrusion detection. In: Proceedings of the Ist USENIX workshop on intrusion detection and network monitoring, Santa Clara, California, USA, pp 51–62

Htun P, Khaing K (2013) Important roles of data mining techniques for anomaly intrusion detection system. Int J Adv Res Comp Eng Tech 2(5):1850–1854

KDD'99 datasets, The UCI KDD Archive Irvine, CA, USA, 1999 [online] http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

Lee W, Stolfo S,and Chan P (1997) Learning patterns from UNIX process execution traces for intrusion detection. In: Proceedings of the AAAI Workshop on AI Models in Fraud and Risk Management, Stanford, pp 50–56

Modi C et al (2013) A survey of intrusion detection techniques in cloud. J Netw Comp App 36:42–57

Mukkamala S, Sung A, Abraham A (2004) Designing intrusion detection systems: architectures and perspectives. Annual review of communications, The Int Eng Consortium (IEC), Chicago, 57:1229–1241

Payne T et al (1997) Experience with rule induction and k-nearest neighbor methods for interface agents that learn. IEEE Trans Knowl Data Eng 9(2):329–335

Rawat S et al (2006) Intrusion detection using text processing techniques with a binary-weighted cosine metric. J Info Assur Security 1:43–50

Tandon G and Chan P (2005) Learning useful system call attributes for anomaly detection. In: Proceedings of the 18th International Artificial Intelligence Research Society Conference, Florida, pp 405–410

Vokorokos L. and Balaz A (2010) Host-based intrusion detection system. In: 14th International Conference on Intelligent Engineering System, Spain, pp 43–47

Warrender C, Forrest S, Pearlmutter B (1999) Detecting intrusions using system calls: alternative data models. IEEE Symposium on Security and Privacy, Oakland, pp 133–145

Wespi A, Dacier M, Debar H (2000) Intrusion detection using variable length audit trail patterns. Recent Adv Intru Det 1907:110–129

Ying L, Yan Z, Jia O (2010) The design and implementation of host-based intrusion detection system. In: Third International Symposium on Intelligent Information Technology and Security Information, Jinggangshan, pp 595–598