A Fingerprinting System Calls Approach for Intrusion Detection in a Cloud Environment

Sanchika Gupta
Department of E&CE
Indian Institute of Technology, Roorkee
Uttarakhand, India
dr.sanchikagupta@gmail.com

Anjali Sardana
Department of E&CE
Indian Institute of Technology, Roorkee
Uttarakhand, India

Padam Kumar
Department of E&CE
Indian Institute of Technology, Roorkee
Uttarakhand, India
padamfec@iitr.ernet.in

Ajith Abraham
IT For Innovations - Center of Excellence
VSB-Technical University of Ostrava, Czech Republic
*Machine Intelligence Reserch Labs (MIR Labs), WA, USA
ajith.abraham@ieee.org

Abstract— **Cloud Computing envisioned as the next generation architecture for IT enterprises, has proliferated itself due to the advantages it provides. Cloud Computing provides solutions for carrying out efficient, scalable and low cost computing. Due to the distributed nature of cloud based system, it is vulnerable to a large category of attacks out of which VM based attacks are most common. To counter these attacks we need Intrusion Detection System (IDS), which is used to monitor network traffic and policy violations from unauthorized users. Anomaly Detection is a technique of Intrusion Detection, which is used to detect intrusions by monitoring system activity and finding out patterns that do not comply with the normal behavior. In this paper an approach for anomaly detection in cloud environment is presented, which is based upon analysis of system call sequences generated by the virtual machines to the hypervisor. Our proposed implementation prevents malicious VM users to modify well known frequently executed programs.**

Keywords: cloud, IDS, anomaly detection, system call, xen, finger print.

I. INTRODUCTION

Cloud Computing has evolved as a major platform that provides a variety of services on a pay per usage model. It provides services at software, platform and application layer. The US National Institute of Standards and Technology (NIST) have captured five essential cloud characteristics: on-demand self-service, ubiquitous network access, resource pooling, rapid elasticity and measured service [1]. Because of the various services it provides and the ease of access to services in a cloud, it is vulnerable to a large number of network and host based attacks [2]. However the concept of Cloud Computing is not new but due to its globalization and enormous usage there is an immediate requirement to look at its security aspects.

Cloud is basically of two types: Compute Cloud and Data Cloud. The Cloud which provides computing power as a service comes under the category of compute cloud while others providing storage services are known as data cloud. Both of these cloud service model have different requirements with respect to their security. For example, securing a Compute Cloud majorly include detection of intrusive system call sequences, malwares, trojans that are targeting to disrupt the computational efficiency provided by the cloud vendor, while securing a data cloud will mainly focus on network based attacks that may try to get access to unauthorized information stored in it.

It is found that a large category of attacks are launched through malicious VM's allocated to the cloud users. Major attacks identified till date includes DDoS attacks from virtual machines (VM's), VM Hoping, VM Rootkit, VM escape etc. Attackers target the Cloud infrastructure for consuming bandwidth, storage capacity and processing power.

In case of network based attacks such as TCP SYN Flooding, malicious VM's generate and send large number of TCP SYN packets to the privileged domain or other neighboring VM's to consume bandwidth, storage and processing power of the end systems and hence will decrease the quality of cloud services. Many other attacks launched from malicious VM's that utilizes network vulnerabilities include buffer overflow attacks, network sniffing etc.

There are other attacks that utilize vulnerabilities in the virtualization environment used for allocation of cloud infrastructure. Such vulnerabilities present in virtualization environment are because of improper configuration of services provided to the cloud users. There has been a lot of research going on to provide security to the virtualization software such as hypervisors [3] so that commonly known VM based attacks that utilizes hypervisors miss-configuration and vulnerabilities can be minimized [4].

It has been noticed that cloud vendors are more interested in increasing the compute power of the clouds with minimization of resources so that the available services can be delivered at low cost and new services can be added to it. Hence any defensive mechanism to tackle well known attacks on cloud can only be taken into practice if it adds low cost to the cloud infrastructure for its deployment, maintenance and operation. Existing solutions such as IDS, IPS which are proposed for securing cloud infrastructure focus more on providing robust security to the cloud users and vendors but

they lack in providing a solution that can be deployed efficiently to the cloud infrastructure [5].

In this paper an approach for anomaly detection in cloud environment is presented which is based upon statistical analysis of system call sequences generated by the virtual machines to the hypervisor. Existing schemes for detection of malicious system calls from virtual machines on cloud infrastructure does not focus on statistical analysis of system call sequences rather focus on integrity of individual system calls. It will help in detecting a malicious system call but not malicious system call sequences which decrease the probability of detecting an intrusive call sequence if present in frequently used system call sequences.

II. RELATED WORK

The research presented in this paper is related to anomaly detection for severity analysis in virtual machines. It is based on the statistical analysis of malicious system call sequences which are generated by the virtual machines. So we have drawn the existing literature in this area so that a comparative analysis could be made with our scheme.

Forest et al. [6] presented a method for anomaly intrusion detection at the process level. Discrimination is made between normal and abnormal characteristics. Normal is defined as short sequences of system calls that are generated by running privileged processes. Using this method they were able to detect various attacks like buffer overflow, symbolic link attack etc.

Lee et al. [7] extended the work of Forest et al. and identified the normal and abnormal patterns in Unix processes. Machine Learning based approach was used to identify misuses and intrusions in UNIX system. They applied RIPPER, a rule learning program to the audit data of UNIX sendmail program.

Warrender et al. [8] proposed a method for detecting intrusions using intrusive system calls. In this the sequence of system calls was identified in the kernel of an operating system. For experimentation they compared 4 methods for observing normal behaviors and detecting intrusions based on system calls in privileged processes. This scheme however is not specific to cloud computing platform.

Ghosh et al. [9] used ANN (artificial neural network) techniques to learn normal sequences of system calls for specific UNIX system programs. More than 150 program profiles were established. For each program, a neural network was trained and used to identify anomalous behaviour. They used DARPA dataset for establishing profiles.

Liao et al. [10] proposed text categorization techniques for intrusion detection. Here instead of the storing short sequences of system calls, frequencies of system calls are used to identify the program behavior using kNN classifier. The kNN classifier is used to classify program behavior as normal or intrusive.

Ye et al. [11] proposed an intrusion detection approach for system call sequences and rule extraction. In this paper, an approach for anomaly intrusion detection is presented and applied to monitor the abnormal behavior of processes. The approach is based on rough set theory and capable of extracting a set of rules with the minimum size to form a normal behavior model from the record of system call sequences generated during the normal execution of a process. It may detect the abnormal operating status of a process. The normal behavior model in terms of the system call sequences is defined. And the detection algorithm is given for the application of rough set theory in intrusion detection. The illustrative example shows that it is feasible and effective.

Bharadwaja et al. [12] proposed Collabra which is integrated with every virtual machine monitor and acts as a filtering layer to detect attacks and prevent illicit access to the VMM and the host. It also performs filtering of malicious hyper calls at the guest OS level before routing the call to the VMM. This scheme however does not looks for intrusive system call sequences which can help in preventing an attack before it is targeted if the system call sequence falls outside any valid system call sequences.

Jin et al. proposed VMFence [13] which is used to monitor network flow and file integrity in real time. But this architecture is more computationally complex as it checks for attack patterns from data coming via all VM's connected to the privileged domain. Also it does not take into account malicious system call sequences from VM's targeted on the cloud infrastructure.

Arshad et al [14] proposed an automatic intrusion diagnosis approach for clouds. In this paper they have analysed a set of three security attributes i.e. availability, confidentiality and integrity. They have categorized all the attacks on the basis of these three security attributes. Then they have identified what kind of attack is being generated by the system calls and mapped those system calls to any one of these attributes. Both supervised and unsupervised methods have been used for preparing the training datasets. However the implementation on virtual machines or real cloud environment is not demonstrated.

Arshad et al. [15] proposed a novel intrusion severity approach for cloud. They are focused on presenting a machine learning based approach which make use of virtual machine specific parameters such as security requirement, SLA state and frequency of attack. They have demonstrated that the proposed approach is effective to address the severity analysis in Cloud. But it does not detect attacks from malicious system call sequences.

III. PROPOSED WORK

According to our studies, we have identified that there is an immediate need of security against malicious insiders and users that can attack cloud environment by manipulating execution of frequently used programs for performing malicious operations over cloud.

We have proposed a new approach to intrusion detection [19-27], which detects malicious system call sequences of well-known programs. The term "well known programs" refers to those programs that are frequent in their execution and whose signature of execution remains fairly constant with respect to time. Our proposed implementation prevents malicious VM users or cloud insiders to modify well known

frequently executed programs. These programs can be those which are allocated to a particular VM User as software or programs that are related with the functioning of VM's such as networking, configuration or operation of the cloud system.

The word program is abstracted here and it can imply any executable instance which runs on VM for providing services. One of the known examples can be of a well-known .exe file provided to VM users that is automatically executed from remote privileged domain when a VM particular to a user starts and connected with the network. As it is stored on the VM a malicious user is always free to remove it and create a new copy of it with same code but malicious code inserted into it. Such malicious code can result into malicious system calls to get executed from the VM that may target to get extra privileges of other virtual machines. An intelligent user may use malicious system call fired from execution of the program to disable some of security controls and policies over its own VM.

Our proposed scheme overcomes the above scenario by keeping an eye over the execution of well-known programs. We in our scheme take and store the finger print of such well known programs in terms of system call sequence arrays and use the same finger print to verify their correctness every time they are fired from a VM. We are different from other schemes that store valid system call sequences for execution of a single or defined set of programs. Our scheme creates individual arrays for each system call that contain immediate next valid system calls, and this array is created from analysis of a set of well-known programs.

For creating such arrays, we have taken system call logs of normal program execution as a base for detection of anomaly and our storage data structure for capturing the finger print of execution of program is in terms of an array. Individual array is created for each system call that comes in the normal execution of the program. The array consists of the System call number of the system call for which it is created with next indices containing the consecutive next system calls that are executed just immediately after it any time during the execution of the well-known program. So for example if we have N different system calls we have N different arrays for each system call each containing next system calls that are fired just after it during execution. For each well-known program whose execution is vulnerable, and manipulations are probable over it, we analyze its execution of system calls and create such fingerprint for it. This finger print for the program is stored and used in future for detection of malicious system calls detection. Finger print is created and stored in both normal and abnormal but authentic conditions of program executions. Hence finger print is assured to contain all the possibilities that can happen under normal program execution conditions. Our detection is signature based as if the finger print of normal program execution does not matches with the signature obtained from system call logs gathered from everyday execution of the program, an abnormal session is detected.

During detection phase whenever the program is executed a system call log is created with the use of system call tracer in privileged domain. From the log same finger print or array's for each system call is created in the same fashion. Then a one to one matching of those finger prints is done. If any mismatches are found an abnormal session gets recorded in the privileged domain and notified to the cloud administrator. The notification can be in terms of messages that contain the system log name in which an anomaly is detected. This will allow cloud admin to trace back the anomalous execution and do the necessary actions over the VM from which such sessions are recorded.

The Finger print creation strategy from system call logs is specified in Figure2. In this Figure system calls are extracted from system call logs. System call logs are obtained with the use of utility programs such as Strace and Ntrace. They are then mapped to their respective system call numbers as present in the mapping file.

The mapping file relates each system call with its unique system call number that is uniquely identifiable and easy to use and remember. The final Sys call log with System call Numbers is used for generating the individual system call array for each system call for a set of well-known program executions. The collection of all such arrays will result in a finger print of normal program executions. This finger print is not created in just one program execution, but it is analyzed and verified in various execution of the a single and set of program so that it is assured that finger point that will then be used for verification does not varies.

This Finger print is stored as a base for anomalous session detection. Now every time this program is executed from VM a system call log is created for the program execution and again the same method is applied for creation of system call array or fingerprint of program execution. This Finger print is matched with the base fingerprint stored for that program in privileged domain if it matches this is not a problem and if it shows mismatches proper notification is propagated to Cloud admin about the anomalous executions so that necessary actions can be taken.

Figure1 depicts how system calls fired from application or program executions are captured from Strace and collected as system logs. These system logs are then looked for individual system call names so that they can be mapped to their corresponding unique system call number according to the mapping provided in the Mapping file. Now for each unique system call as obtained from system call logs an array is created which stores the system call numbers of the immediate system calls that are called after it in the program execution. This array is not only for a single program execution.

This finger print is not only for a single program execution but this will store signature of valid system call sequences for a set of well-known programs. Here by well-known program we mean those programs which are vulnerable to malicious code insertions that can change the order of system call execution of that program. Hence the finger print created will inform for any subversion in the normal program executions. Figure2 explains how the system call finger print of a program execution is compared with finger print of set of well-known programs stored in finger print database for ensuring its validity. In example as all the arrays values matches with the values present in finger print database, it signifies that the execution sequence of system call is valid and is not subverted.

IV. IMPLEMENTATION DETAILS

For showing a demo implementation we have installed a system call logger for windows called Ntrace [16]. Similarly a system call logger for Linux environment called Strace [17] has been also used in our prototype implementation and analysis. These program logs the system calls fired during a program execution. Virtual machines are assigned to users which fire commands to be executed by the hypervisor and their system call logs are stored in the privileged domain Dom-0 in xen [18]. We collect such system call logs for each program or executable by running them and then looking for the system calls which are executed by them.

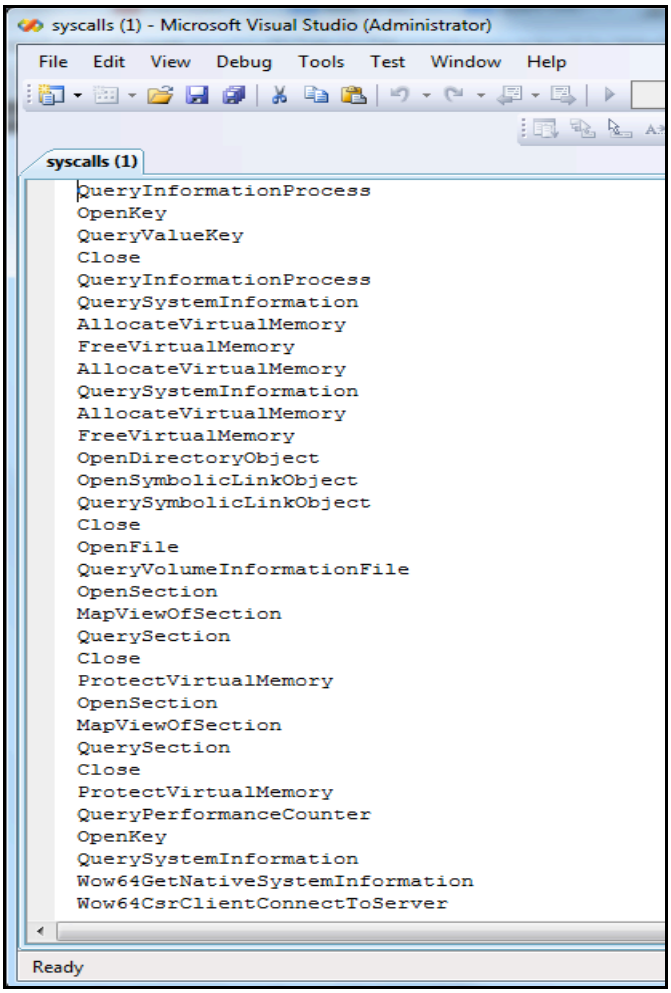

Figure1. Snapshot of System call log from a Program Execution

The system call pattern of a particular well known program remains same always and varies in some defined way in valid abnormal conditions. By system call pattern we mean the sequence of system calls with their number and calling order, fired by the program to the underlying operating system. From the system call logs we gather all the individual unique system calls which are then mapped to their system call numbers. If for that system call the array already exists in the signature database we do not create a new array for it however if that system call is a new system call in the database we will create a new array for it. Whether the array is present or absent the immediate next system call following that system call are appended to that array in the next available index location. The finger print database remains same and also the array of the individual unique system call numbers. When a new program is executed for analysis the system calls are extracted and the entries of immediate next system calls are made in the same database in their corresponding array location. If the entry already exists it is not entered again however if it is not there the new entry is made in the next available index for a particular system call.

In this way the finger print database is created for a set of well-known programs executions. But the analysis of each program execution is done several times in different conditions to ensure that the execution signature does not vary. Figure 1 gives a snapshot of system calls log of a particular program execution from which unique system calls are extracted for finger print database creation and updation.

The system call logs for a particular application are analyzed and individual system calls are extracted. These are then mapped to a sys call number (Sys No) via the mapping file which is stored in Dom-0. During detection the same procedure is followed for creation of individual array for individual program execution from their system call logs collected. The finger print is created in the same manner. For each system call an array is created that contains the immediate next system calls executed after it. The array is then looked into finger print database for matching of the system calls if such a match occurs then it represents that the signature of execution is valid and there is no subversion in program execution.

The fingerprint as explained in Figure 2 is then created for the program execution. But the finger print is created after analysis for a particular program in its normal and abnormal executions. We have analyzed various program executions for various executable to verify that system calls for normal execution of a program does not frequently varies in practice. We have developed the Fingerprint generation technique as proposed in the architecture however its efficient deployment over cloud and results evaluation is currently in progress.

V. CONCLUSIONS

In this paper, an approach for detection of malicious program executions in Cloud environment is proposed, which is based upon analysis of system call sequences generated by the virtual machines programs to the hypervisor. Our proposed implementation prevents malicious VM users to modify well known frequently executed programs. Hence it detects any malicious code insertions in frequent and well known program executions. This approach of modelling valid system call sequences helps in detection of system call sequences which are anomalous and falls outside the set of valid system call sequences hence result in anomaly detection. This approach can help in the detection of subverted operations that may be launched by an attacker from a VM to exploit the vulnerability of a particular operating system or computing platform.

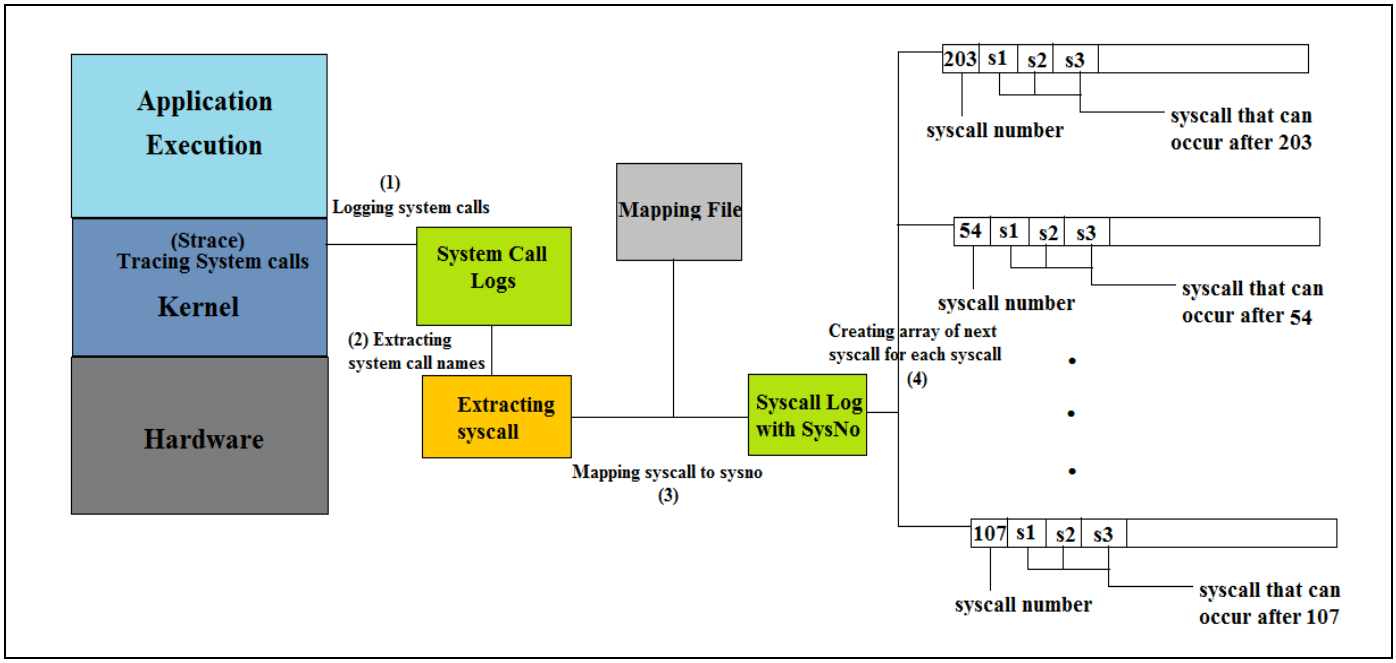

Figure 2. Finger print creation from System call logs

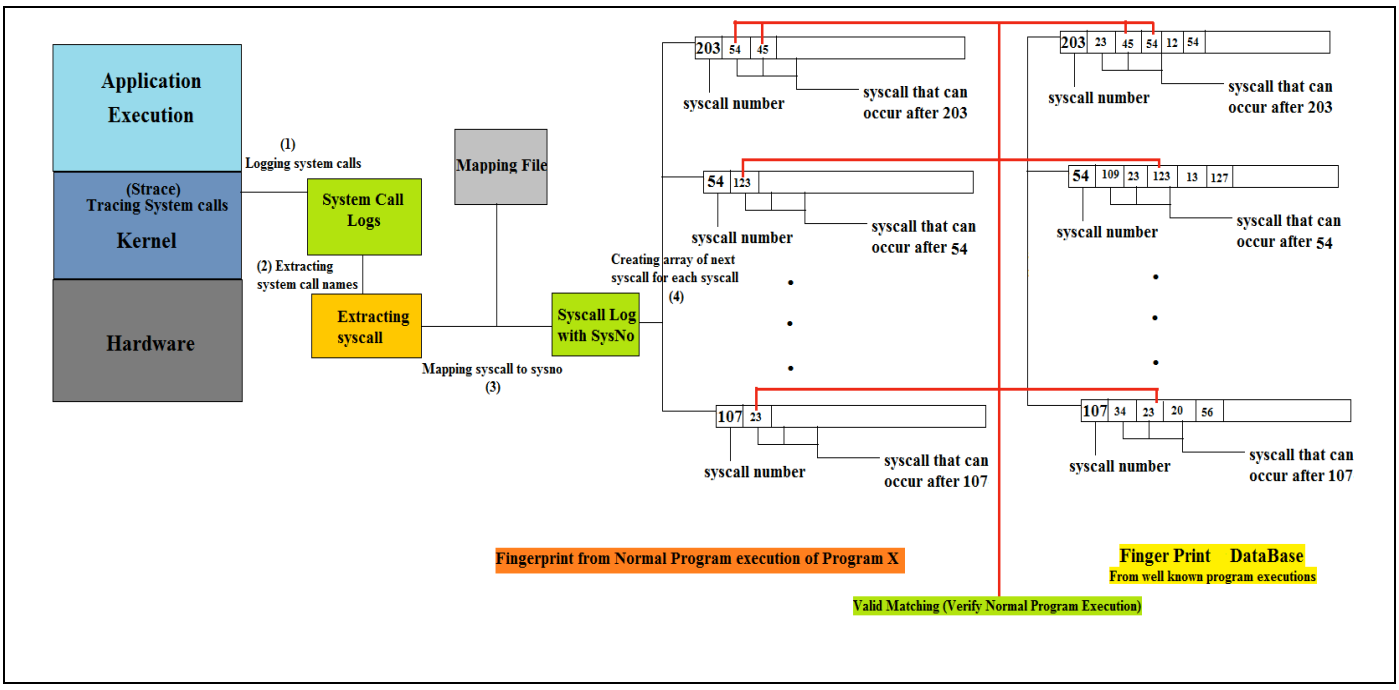

Figure3. Detecting valid program executions from Finger print

We have verified the initial design of the proposed solution and identified that it is low complex as only a single finger print database will be required for verification and validation of execution of multiple programs. We have deployed a prototype implementation of the same however the detailed implementation in real time environment with scalability and adaptability issues is in study. The future work will be hence focused on validating the detailed implementation of it on a commercial cloud environment.

REFERENCES

1. E. Brown, "NIST Issues Cloud Computing Guidelines for Managing Security and Privacy," *National Institute of Standards and Technology Special Publication 800-144*, 2012.
2. P.M.T. Grance, "Effectively and Securely Using the Cloud Computing Paradigm (v0.25)," 2009; http://csrc.nist.gov/organizations/fissea/2009-conference/presentations/fissea09-pmell-day3_cloud-computing.pdf.
3. S. Gupta, S. Horrow, and A. Sardana, "IDS Based Defense for Cloud Based Mobile Infrastructure as a Service," *Services (SERVICES), 2012 IEEE Eighth World Congress on*, IEEE, pp. 199-202.
4. B. Grobauer, T. Walloschek, and E. Stocker, "Understanding Cloud Computing Vulnerabilities," *Security & Privacy, IEEE*, vol. 9, no. 2, pp. 50-57.
5. S. Gupta, S. Horrow, and A. Sardana, "A Hybrid Intrusion Detection Architecture for Defense against DDoS Attacks in Cloud Environment
Contemporary Computing," *Contemporary Computing*, Communications in Computer and Information Science 306, Springer Berlin Heidelberg, 2012, pp. 498-499.
6. A.H. Steven, F. Stephanie, and S. Anil, "Intrusion detection using sequences of system calls," *J. Comput. Secur.*, vol. 6, no. 3, 1998, pp. 151-180.
7. Wenke Lee, Philip K. Chan, "Learning Patterns from Unix Process Execution Traces for Intrusion Detection," *Proceedings of AAAI97 Workshop on AI Methods in Fraud and Risk Management*, pp. 50-56.
8. S.F. Christina Warrender , Barak Pearlmutter, "Detecting Intrusions Using System Calls: Alternative Data Models," *In IEEE Symposium on Security and Privacy*, IEEE.
9. A.S. Anup K. Ghosh, Michael Schatz, "Learning Program Behavior Profiles for Intrusion Detection," *Proceedings of 1st USENIX Workshop on Intrusion Detection and Network Monitoring*.
10. V.R.V. Yihua Liao, "Using Text Categorization Techniques for Intrusion Detection," *Proceedings of the 11th USENIX Security Symposium*.
11. Q. Ye, X. Wu, and B. Yan, "An Intrusion Detection Approach Based on System Call Sequences and Rules Extraction," *e-Business and Information System Security (EBISS), 2010 2nd International Conference on*, pp. 1-4.
12. S. Bharadwaja, S. Weiqing, M. Niamat, and S. Fangyang, "Collabra: A Xen Hypervisor Based Collaborative Intrusion Detection System," *Information Technology: New Generations (ITNG), 2011 Eighth International Conference on*, pp. 695-700.
13. H. Jin, G. Xiang, D. Zou, S. Wu, F. Zhao, M. Li, and W. Zheng, "A VMM-based intrusion prevention system in cloud computing environment," *The Journal of Supercomputing*, 2011, pp. 1-19.
14. J. Arshad, P. Townend, and J. Xu, "An automatic intrusion diagnosis approach for clouds," *International Journal of Automation and Computing*, vol. 8, no. 3, 2011, pp. 286-296.
15. Junaid Arshad, Jie Xu, "A novel intrusion severity analysis approach for Clouds," *Future Generation Computer Systems, The International Journal of Grid Computing and eScience*, vol. 28, no. 7, 2011, pp. 965-1154.
16. Roger, "NtTrace - Native API tracing for Windows," 2012; http://www.howzatt.demon.co.uk/NtTrace/.
17. "Strace," 2008; https://wiki.ubuntu.com/Strace.
18. "XEN," 2012; http://www.xen.org/.
19. Alvaro Herrero, Emilio Corchado, Maria Pellicer and Ajith Abraham, MOVIH-IDS: A Mobile-Visualization Hybrid Intrusion Detection System, Neurocomputing Journal, Elsevier Science, Netherlands, 72(15), pp. 2775-2784, 2009.
20. Ajith Abraham, Crina Grosan and Carlos Martin-Vide, Evolutionary Design of Intrusion Detection Programs, International Journal of Network Security, Vol.4, No.3, pp. 328-339, 2007.
21. Yuehui Chen, Ajith Abraham and Bo Yang, Hybrid Flexible Neural Tree Based Intrusion Detection Systems, International Journal of Intelligent Systems, John Wiley and Sons, USA, Volume 22, pp. 1-16, 2007.
22. Sandhya Peddabachigari, Ajith Abraham, Crina Grosan and Johnson Thomas, Modeling Intrusion Detection System Using Hybrid Intelligent Systems, Journal of Network and Computer Applications, Elsevier Science, Volume 30, Issue 1, pp. 114-132, 2007.
23. Ajith Abraham, R. Jain, J. Thomas and S.Y. Han, D-SCIDS: Distributed Soft Computing Intrusion Detection Systems, Journal of Network and Computer Applications, Elsevier Science, Volume 30, Issue 1, pp. 81-98, 2007.
24. S. Chebrolu, Ajith Abraham and Johnson Thomas, Feature Deduction and Ensemble Design of Intrusion Detection Systems, Computers and Security, Elsevier Science, Volume 24/4, pp. 295-307, 2005.
25. S. Mukkamala, A. Sung and Ajith Abraham, Intrusion Detection Using Ensemble of Soft Computing and Hard Computing Paradigms, Journal of Network and Computer Applications, Elsevier Science, 28(2), pp. 167-182, 2005.
26. Ajith Abraham, C. Grosan and Y. Chen, Cyber Security and the Evolution in Intrusion Detection Systems, Journal of Engineering and Technology, ISSN 0973-2632, I-Manager Publications, Vol. 1, No. 1, pp. 74-81, 2005.
27. S. Mukkamala, A. Sung, Ajith Abraham and Vitorino Ramos, Intrusion Detection Systems Using Adaptive Regression Splines, Enterprise Information Systems VI , Seruca, I.; Cordeiro, J.; Hammoudi, S.; Filipe, J. (Eds.) Springer-Verlag, ISBN: 1-4020-3674-4, pp. 211-218, 2006.