Natural Language Processing based Anomalous System Call Sequences Detection with Virtual Memory Introspection

Suresh K. Peddoju[1], Himanshu Upadhyay[2]
Applied Research Center
Florida International University
Florida, USA

Jayesh Soni[3], Nagarajan Prabakar[4]
School of Computing and Information Sciences
Florida International University
Florida, USA

***Abstract*—Malware has become a significant problem for the security of computers in this scientific era. Nowadays, machine learning techniques are applied to find anomalous activities in computers especially in virtualization environments. Identifying anomalous activities in virtual machines with virtual memory introspector and analyzing data with machine learning techniques are need of current trend. In this paper, an anomaly detection method is implemented using Natural Language Processing (NLP) based on Bags of System Calls (BoSC) for learning the behavior of applications on Windows virtual machines running on Xen hypervisor. During this process, system call traces are extracted from normal applications (benign processes) and malware affected applications (malicious processes) with the help of virtual memory introspection. Preprocessing of extracted system call sequences is done to obtain valid system call sequences through filtering and ordering of redundant system calls. Further, analysis of behavior of system call sequences is carried out with NLP based anomaly detection techniques. During this process, Cosine Similarity Algorithm (Co-Sim) is applied to identify malicious processes running on a VM. Apart from this, Point Detection Algorithm is applied to precisely locate the point of compromise in the system call sequences. The results shown in this paper indicates that both of these algorithms detect anomalies in the running processes with 99% accuracy.**

Keywords*—System call sequence; anomaly detection; natural language processing; memory forensics; cosine similarity

I. INTRODUCTION

Nowadays, virtualization is playing a vital role in distributed systems. It became popular due to its usage and applicability. The significant advantage of virtualization is to provide vast resource sharing, load balancing, and protecting system resources. With the development of virtualization technologies, hypervisor-based methods have evolved to scan virtual machines (VM) and identify the threats happening on it. In the current market, the latest malware is more sophisticated and robust so that no malware detection techniques are capable of detecting and protecting the virtual machine. Thus, many organizations are facing cyber threats to their data and resources. Hypervisor-based malware detection techniques overcome these problems in comparison to host-based malware detection techniques. Virtual Machine Introspection (VMI) is the most versatile malware detection technique to monitor and analyze cyber threats on virtual machines [1][2][3]. VMI is a technique to control the virtual machine run-time state at the hypervisor level, and it is used for forensic analysis of VM activities.

In hypervisor-based environment, it is important to observe virtual machine activities through hypervisor to keep track of benign and malicious activities happening on it. Memory forensics is good technique to extract and analyze memory activities. In this paper, we built a memory forensics architecture which uses VMI. All memory data structures are extracted (including system call sequences) to monitor anomalous activity in VM.

One of the techniques to identify the anomalous behavior of VM is to trace system call sequences of all running applications on VM. Hypervisor will extract system call sequences from memory of VM in runtime. Anomaly detection techniques are applied on collected data to find any anomalies in system call sequences by comparing benign and malicious data. This process will help in identifying the compromised VM on hypervisor. One of the efficient approaches for anomaly detection is Bag of System Calls (BoSC). Kang et al. in 2005 [4] introduces it as a frequency-based technique. According to this method, system call sequences Si are represented as a list {C1, C2, . . . , Cn}, where in n is the number of unique system calls, and Ci is count of system calls, present in the generated input sequence of system calls.

In this paper, we study the richness of using BoSC technique to detect malicious behavior at the process level in a hypervisor based environment. Further, we also propose an algorithm that detects anomalies at a particular point of time using cosine angle similarity. The results shows that considering the sequence of system call occurrences is powerful for detecting real-time anomalies in running processes on Xen hypervisor.

The outline of the paper is as follows: Section 2 describes state of the art related to proposed techniques. The subsequent section provides a system overview. Section 4 discusses the system call feature extraction and pre-processing. In the next section, we explain the proposed algorithm. Furthermore, we give an in-depth explanation of the environmental setup in section 6. The results of the proposed algorithms are presented in section 7. Finally, we conclude the paper in section 8.

II. Related Work

Classifying malware in any production system is of crucial importance for the security of its software components. Static analysis and dynamic analysis are two types of different malware analysis methods. Due to an increase in malware threats, there is a substantial increase in research work on malware detection.

In the static analysis method, we directly analyze source files without executing them [5]. Masud et al. [6], extracted 4-gram byte codes with five different static features of assembly instructions and combined them. For malware detection, they used two classification algorithms, namely decision tree algorithm and support vector machine. Ye et al. [7] used an association mining algorithm that generated association rules by developing an Intelligent Malicious code Detection System (IMDS) to obtain import function information. Finally, they used an association rule-based classification algorithm to detect malware.

However, techniques such as encryption, packing of malware, and polymorphism affect static based anomaly detection methods. Analyzing the behavior of an application is known as dynamic analysis. Its basic idea is to analyze the execution of the application [8]. This approach solves many of the problems of static-based analysis.

Many authors have used Hidden Markov Model (HMM) based classifier to detect anomalies in system calls [9][10][11][12][13][14]. However, each author uses a different set of techniques for improving the precision of anomaly detection. Alarifi and Wolthusen [15] took sequences from a virtual machine and then trained them using HMM. Their HMM-based method gave fewer detection rates since it required fewer training samples. The detection rate was 97% by using 780k system calls for training. Wang et al. [11] used the probability score and threshold value of the whole sequence. Cho et al. [13] used HMM by training regular user-level privilege operations. Hoang et al. [14] introduced an anomaly detection technique for multi-layer by using the sliding window approach. Warrender et al. [9] provide a comparison of STIDE [16], RIPPER [17], and HMM-based methods. These methods had different performance characteristics, while HMM performed with good accuracy. However, HMM requires multiple passes through the training data, high computational power, and needs large storage, especially for significant sequence length. Time series based modeling has been performed in [18][19]. The Kernel State Modeling (KSM) technique uses sequences of system call sequences as an individual task of kernel modules [20]. This method calculates the probability of occurrences of the finite number of states in malicious traces of system calls and compares against the expectations of normal traces. The KSM results in higher detection rates in comparison to HMM-based methods for UNM dataset. For feature extraction, neural-net based embedding is used for single dimensions data [21][22][23][24]. Suresh et al. [25][26] introduce machine learning algorithms for feature extraction for multidimensional data.

III. System Overview

The proposed framework and methodology is described in this section. This framework describes how system call sequences are extracted and analyzed by using a VMI based architecture and machine learning methods. This workflow collects system call traces of running processes and introspects the malicious behavior of processes on guest VM. The following subsections describe the architecture, methodology, and procedure to create custom malware.

A. Architecture

The architecture of the proposed memory forensic framework, as shown in Fig. 1, consists of four modules: the Virtualization module, the Advanced Cyber Analytics module, the Malware repository module, and the Test Control Center module. The proposed framework acquires smart memory introspection features, analyzes them with advanced cyber analytics algorithms along with a control center for managing the system for visualizing the results.

The following sub-sections describe the functionality of individual modules and their components.

1) Virtualization: In this module, smart memory introspection is performed on Virtual Machine (VM) using VMI API to introspect and perform memory forensics. This module consists of different sub-modules such as Introspector and Security Agent.

a) Introspector: This module extracts low-level data from the memory of virtual machines running on a hypervisor, and transfers this data to agent listener(s) for anomalyanalysis. The Introspector interfaces with hypervisors to ensure that the state of the virtual machines (running, stopped, or shut-down) can be manipulated, and VMs can be added and deleted as needed.

b) Security Agent: This sub-module initiates scans on VMs using the LibVMI library to perform introspection. Its primary mechanism is to extract data from a VM and send the data to the agent listener for further analysis. The Security Agent has various features that allow the agent to scan processes, invariant data structures, and to monitor files changes.

Fig. 1. System Call Traces with Virtual Machine Introspection.

2) Advanced cyber analytics: This module comprises of different machine learning and deep learning algorithms to train the model and perform a test on that model for further prediction and analysis of data. The baseline data is considered as benign data, and the test vector injected data is known as malicious data. The data extracted by using the introspection module is stored on a database server and then analyzed using different cutting-edge machine learning techniques.

3) Malware repository: This repository consists of a massive set of malware that compromises kernel-level data structures. This repository includes different malware for Windows and Linux. This malware repository also consists of custom malware sets to compromise the specific context of kernel data structures.

4) Test control center: With the help of the Test Control Center module, the operator can control and manage the whole framework and its modules with a user interface. The operator can handle the VM operations, such as creation, deletion, stop, start, pause, and view. Also, the operator can control the VMs by installing or running malware and benign applications. The operator visualizes the processed results from the Advanced Cyber Analytics module for further analysis.

B. Methodology

In the current implementation of this framework, system call traces are collected from live VM using Virtual Memory Introspection method. An Introspector package developed on hypervisor which consists of two modules *introspector* and *security agent*. Among these modules, introspector module gets connected with the VM and initiates the security agent module to extract the system call traces from live memory of VM. Further, security agent sends extracted data to database with the help of other application called *Agent Listener*. This application intern stores information into database. Next step is to pre-process and analyze the collected system call traces using anomaly detection algorithms. In view of these, a custom application is designed to manage the VM, initiate the scanning, view results and many more. For further study, an operator can process these traces.

C. Custom Malware

A set of custom malware were created to compromise system call sequences by way of DLL injection. This injection hooks into the write function of processes and initiates additional system calls by creating a hidden file on disk. This set of custom malware is used in experiments to compromise system call sequences.

IV. Feature Extraction Technique

A process behavior is defined with an approach based on angle similarity. As part of this method, the occurrences of system calls generated by the process are considered, instead of the temporal ordering of system calls. This paper presents a technique called angle similarity which is similar to text classification for anomaly detection, where a sequence of system calls is considered as the document, and individual system calls are viewed as a word. The system-call sequence are extracted under normal operation are collected from the hypervisor. Fig. 2 shows the sample sequence of system calls.

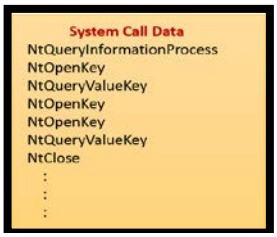

Fig. 2. Sample Sequence.

According to this approach, each and every system call is mapped to a unique number from 0 to 450 to a given sequence of the system calls. The total unique system calls for Windows is 450. A sample mapping of system calls is shown.

System Call Name	NtQueryInformationProcess	NtOpenKey	NtQueryValueKey	NtOpenKey	NtQueryValueKey	NtClose
Mapping Number	25	18	23	18	23	15

We create a Bag of System Calls of 450 dimensions where each cell value designates the frequency of the i^{th} system call. The following Fig. 3 shows a sample Bag of System calls:

	0	1	2	3	4	5	6	7	8	9	...	440	441	442	443	444	445	446	447	448	449
0	1	0	0	0	6	10	17	0	0	0	...	0	0	0	0	0	0	0	0	0	0

Fig. 3. BoSC of 450 Dimensions.

V. Detection Algorithm

The proposed approach computes the cosine similarity between the features from normal processes and malicious processes. Cosine similarity is a similarity measure between two vectors that calculate the cosine angle between them.

The cosine angle between two vectors is calculated using their Euclidean dot product. Equation 1 shows the Euclidean dot product.

$$A.B = \|A\|\|B\| \cos\theta \tag{1}$$

Given two vectors of n dimensions, *A* and *B*, the cosine similarity value is calculated as the function of cos(θ) shown in Equation 2:

$$Similarity = cos(\theta) = \frac{A.B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{2}$$

Where A_i and B_i are the features of vectors A and B respectively in the equation.

A. Anomaly Detection Algorithm

The following algorithm detects anomalies in the running processes in Windows VM on Xen hypervisor.

For a given set of processes in baseline and test data, use its system-call sequences and mapping table to map system-call name to number. An anomalous system call sequences can be detected by using Algorithm #1, which is shown.

B. Point Detection Algorithm

A Point detection algorithm detects a particular point in the process execution where the malicious attack has happened.

Sequence length is the number of the system calls taken into consideration. Sequence length of the system call is provided as input to the Point Detection Algorithm as given below in Algorithm #2, BoSC of an anomalous process from the above anomaly detection algorithm #1, and BoSC of a normal process.

For point detection algorithm, we use a sliding window of varying lengths and calculate the cosine similarity for that particular window. If the cosine similarity is less than 0.99, then that process within that window is considered as anomalous. Fig. 4 depicts the point detection method.

Algorithm 1: Anomaly detecting process for system call sequences.

Algo #1	***Anomaly Detection Algorithm***
Step 1:	**for** each process B($\mathcal{P}_i$) in Baseline { $\mathcal{P}_1, \mathcal{P}_2 \mathcal{P}_3, \ldots\ldots, \mathcal{P}_n$ }**do** Map S_i ->S_{num} using the System Call mapping table. Where S_i is system call name and S_{num} is System call mapping number Create bag-of-words for $\mathcal{P}_i$ by counting occurrences of each S_i
Step 2:	**for** each process in T($\mathcal{P}_i$) the TestData{ $\mathcal{P}_1, \mathcal{P}_2 \mathcal{P}_3, \ldots\ldots, \mathcal{P}_n$ }**do** Prepare combined list C($\mathcal{P}_i$) from B($\mathcal{P}_i$)∩T($\mathcal{P}_i$)
Step 3:	**for** each process C($\mathcal{P}_i$) in combined list { $\mathcal{P}_1, \mathcal{P}_2 \mathcal{P}_3, \ldots\ldots, \mathcal{P}_n$ }**do** **if** C($\mathcal{P}_i$) ≠ Mapping table *M(t)***then** P is Anomalous; **else** Map S_j ->S_{num} using the System Call mapping table. Create bag-of-words for $\mathcal{P}_j$ by counting occurrences of each S_j Similarity = f(B($\mathcal{P}_i$), B($\mathcal{P}_j$)) **if** Similarity < 0.99 **then** P is Anomalous; **else** P is Normal; **end do** **end do**

Algorithm 2: Point detection for system call sequences

Algo #2	***Point Detection Algorithm***
Step 1:	*for each Anomalous Process A($\mathcal{P}_i$)and Baseline process B($\mathcal{P}_i$)do* *for each i in range [length(BoSC ofA($\mathcal{P}_i$) – (sequence_length)] do* *Seq(B($\mathcal{P}_i$) = BoSC_of_B($\mathcal{P}_i$)[i:i+sequence_length]* *Seq(A($\mathcal{P}_i$) = BoSC_of_A($\mathcal{P}_i$)[i:i+sequence_length]* *SimVal = Cosine_Similarity(Seq(B($\mathcal{P}_i$), Seq(A($\mathcal{P}_i$))* *if SimiVal< 0.99 then* *Return A($\mathcal{P}_i$), Seq(A($\mathcal{P}_i$)* *end do* *end do*

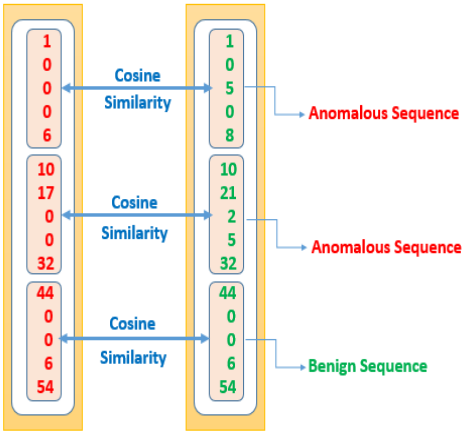

Fig. 4. Point Detection Method.

VI. Environment Setup

The proposed framework is developed on Xen 4.12 hypervisor and managed virtual machines (VM) with Libvirt 5.4.0 library. For getting memory addresses of running processes virtual machine introspector method are being imposed with latest version of DRAKVUF library. The current implementation of this framework consists of two modules namely *Introspector* and *Security* Agent. These modules extract system call traces by inspecting the VM called System Under Test (SUT) using the LibVMI library on top of DRAKVUF in combination with a rekall profile of Google. This rekall profile is files in JSON that ccomprises of memory mappings and offsets of windows data structures. The above two specified modules, are written in Go Language to process the request and extract the system call traces from VM with LibVMI functions. LibVMI library services and the Libvirt library are usedto create, start, or stop virtual machines of windows. An applicationis designed for operator to extract the system call traces. This Application is written in Microsoft Visual Studio .NET framework and comprises of user-defined API calls for introspector communication and other related function calls. An agent transmits eextracted data to the database server. Finally, the stored data is analyzed using different machine learning algorithms. The whole experimental setup is shown in Fig. 5.

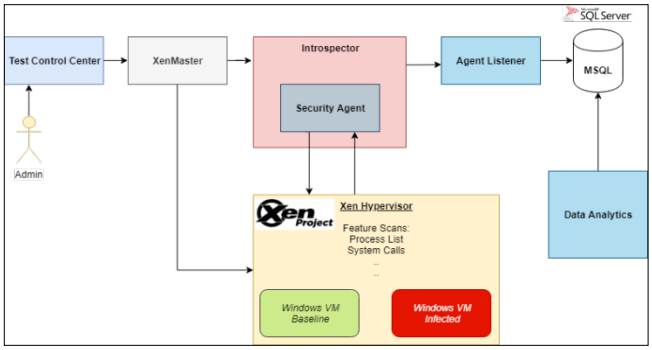

Fig. 5. Experimental Framework for Extracting System Call Traces.

VII. Results

In this section, we present the results of the proposed algorithms.

A. Anomaly Detection Algorithm

We evaluated this algorithm with system-call traces of 1,000,000 system calls with multiple experiments. The total number of unique system calls in Windows operating system is 450. Fig. 6 and 7, display the top 5 system call with their frequencies of a normal SUT application and a malicious SUT application, respectively.

The result shown in Fig. 6 and 7 clearly differentiates between malicious SUT application system call frequencies in comparison with benign SUT application. The following Table \I shows the similarity score between malicious and normal SUT application.

From Table I, we can say that the cosine similarity of a normal SUT Applications is higher whereas malicious SUT application is lower in compared with normal SUT application.

Furthermore, the cosine similarity value is independent of the number of records. Fig. 8 demonstrates this characteristic. We observe the same cosine similarity behavior even with the varying number of records.

B. Point Detection Algorithm

For the point detection algorithm, we tested with a sequence length of 3, 5, 10, and 15. From Fig. 9, we observe that sequence length of 5 gives an ideal cosine similarity value for a single scan.

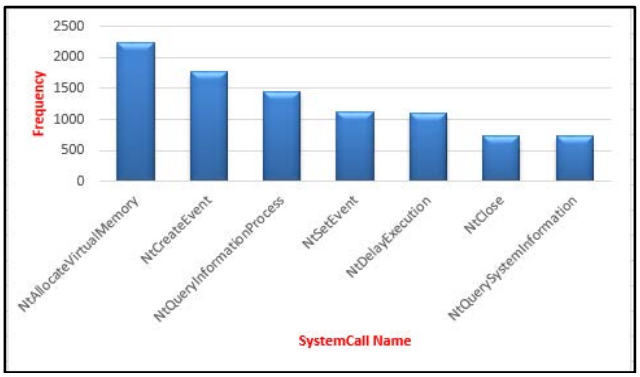

Fig. 6. Top 5 System Calls based on their Frequency of a Normal SUT Application.

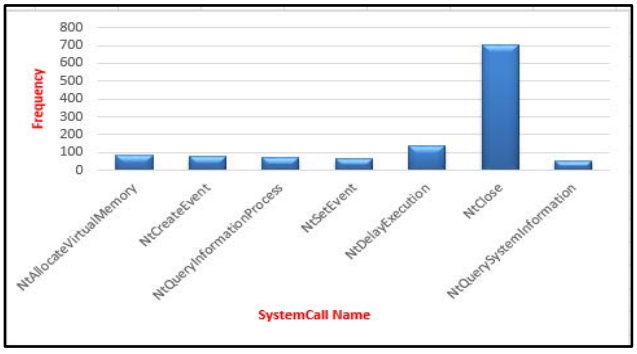

Fig. 7. Top 5 System Calls based on their Frequency of a Malicious SUT Application.

TABLE I. Cosine Similarity between Normal SUT and Malicious SUT Applications

Applications	Normal SUT Application	Malicious SUT Application
Similarity Score	0.99	0.20

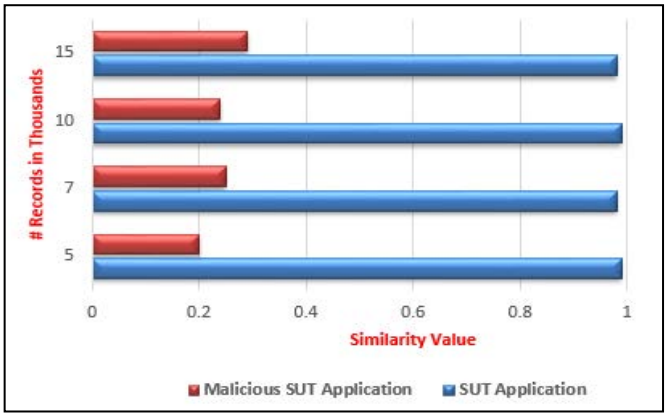

Fig. 8. Cosine Similarity Value for Normal SUT and Malicious SUT w.r.t # Records in Ten Thousand.

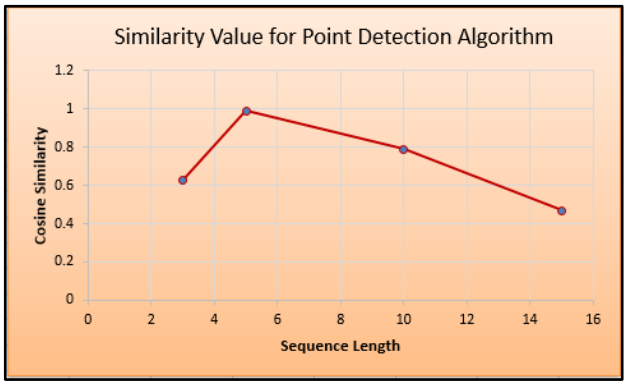

Fig. 9. Cosine Similarity Value w.r.t Sequence Length for a Single Scan.

Furthermore, we evaluated the algorithm with varying scan times. From Fig. 10, we found that with a sequence length of 5, the cosine similarity value is consistently higher in comparison with all other sequence lengths with varying scan times.

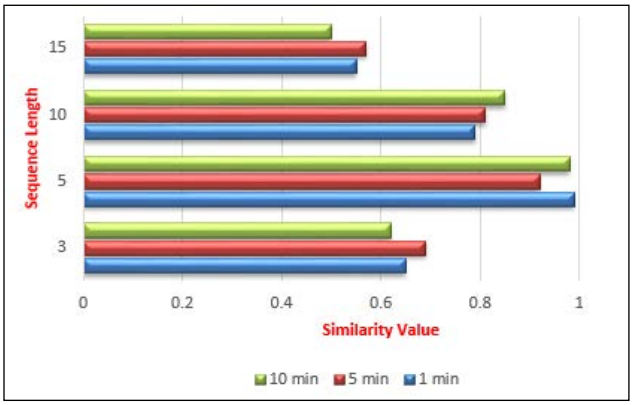

Fig. 10. Cosine Similarity Value w.r.t Sequence Length and Varying Scan Times.

VIII. Conclusions

All intrusion-based detection algorithms work on the hypothesis that regular activities differ from irregular events (intrusions). Anomaly detection algorithms learn a program's behavior. The behavior is in the form of the frequency of system calls raised by the processes under evaluation. We presented two anomaly detection algorithms. Both algorithms calculate the cosine similarity between the processes under examination based on the frequency of system calls. Anomaly Detection Algorithm detects anomaly between benign and malicious system call sequences whereas point detection algorithm detects the timeframe of the malicious attack in the anomalous process. With the help of both of these algorithms we can able to detect malicious behavior of system call sequences with 99% accuracy rate.

Acknowledgment

This work was funded by TRMC of DoD. We are very much thankful for providing facilities and infrastructure to do our experiments. We thank all who directly and indirectly helped us in doing this experiments and results.

References

[1] M H Ligh, A Case, J Levy, A Walters. "The Art of Memory Forensics," 2014.

[2] Xen Project available at https://www.xenproject.org/, 2013.

[3] Hizver, Jennia, and Tzi-ckerChiueh. "Real-time deep virtual machine introspection and its applications." ACM SIGPLAN Notices. Vol. 49. No. 7. ACM, 2014.

[4] D. Fuller and V. Honavar, "Learning classifiers for misuse and anomaly detection using a bag of system calls representation," in Proceedings of the Sixth Annual IEEE Systems, Man and Cybernetics (SMC) Information Assurance Workshop. IEEE, 2005, pp. 118–125.

[5] Payet E, Spoto F. Static analysis of Android programs [J]. Information and Software Technology,2012,54(11): 1192-1201.

[6] Masud M. M., Khan L, Thuraisingham B. A scalable multi-level feature extraction technique to detect malicious executables[J]. Information Systems Frontiers, 2008, 10(1):33-45.

[7] Ye Y, Wang D, Li T, et al. IMDS: Intelligent malware detection system[C]//Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2007: 1043-1047.

[8] Egele M, Scholte T, Kirda E, et al. A survey on automated dynamic malware-analysis techniques and tools[J]. ACM computing surveys (CSUR), 2012, 44(2): 6.

[9] C. Warrender, S. Forrest, and B. Pearlmutter, "Detecting intrusions using system calls: alternative data models," in Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on, 1999, pp. 133–145.

[10] Soni, J., Prabakar, N., and Upadhyay, H. (2019) "Deep Learning approach to detect malicious attacks at the system level: poster," Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks. ACM, pp. 314-315.

[11] W. Wang, X.-H. Guan and X.-L. Zhang, "Modeling program behaviors by hidden Markov models for intrusion detection," in Machine Learning and Cybernetics, 2004. Proceedings of 2004 International Conference on, vol. 5. IEEE, 2004, pp. 2830–2835.

[12] D.-Y. Yeung, and Y. Ding, "Host-based intrusion detection using dynamic and static behavioral models," Pattern Recognition, vol. 36, no. 1, pp. 229–243, 2003.

[13] S.-B. Cho and H.-J. Park, "Efficient anomaly detection by modeling privilege flows using a hidden Markov model," Computers and Security, vol. 22, no. 1, pp. 45 – 55, 2003.

[14] X. D. Hoang, J. Hu, and P. Bertok, "A multi-layer model for anomaly intrusion detection using program sequences of system calls," in Proc. 11th IEEE Intl Conf. Networks, 2003, pp. 531–536.

[15] Alarifi and Wolthusen "Anomaly detection for ephemeral cloud IaaS virtual machines," in Network and system security. Springer, 2013, pp. 321–335.

[16] S. Forrest, S. Hofmeyr, A. Somayaji, and T. Longstaff, "A sense of self for Unix processes," in Proceedings of the 1996 IEEE Symposium on Security and Privacy, May 1996, pp. 120–128.

[17] W. Lee and S. J. Stolfo, "Data mining approach for intrusion detection," in Usenix Security, 1998.

[18] Soni, J., Prabakar, N. and Kim, J-H. (2017) "Prediction of Component Failures of Telepresence Robot with Temporal Data." 30th Florida Conference on Recent Advances in Robotics

[19] G. S. Thejas, J. Soni, K. Chandna, S. S. Iyengar, N. R. Sunitha, and N. Prabakar. 2019. Learning-Based Model to Fight against Fake Like Clicks on Instagram Posts. In SoutheastCon 2019. Huntsville, Alabama, USA. In press.

[20] S. S. Murtaza, W. Khreich, A. Hamou-Lhadj, and M. Couture, "A host-based anomaly detection approach by representing system calls as states of kernel modules," in Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on. IEEE, 2013, pp. 431–440.

[21] Soni, J., Prabakar, N., and Upadhyay, H. (2019) "Feature Extraction through Deepwalk on Weighted Graph,"Proceedings of the 15th International Conference on Data Science (ICDATA'19), Las Vegas, NV, 2019.

[22] Soni, J., Prabakar, N. (2018) "Effective Machine Learning Approach to Detect Groups of Fake Reviewers," Proceedings of the 14th International Conference on Data Science (ICDATA'18), Las Vegas, NV, 2018.

[23] P. Suresh Kumar and S. Ramachandram, "Fuzzy based Integration of Security and Trust in Distributed Computing," Proc of Springer 7th International Conference Soft Computing for Problem Solving (SocProS'2017), Indian Institute of Technology, Bhubaneswar, December 2017.

[24] P. Suresh Kumar, HimanshuUpadhyay and ShekarBansali, "Health Monitoring with Low Power IoT Devices using Anomaly Detection Algorithm," IEEE conference SLICE-2019, Rome, Italy, June 2019.

[25] P. Suresh Kumar and Pranavi S, "Performance Analysis of Machine Learning Algorithms on Diabetes Dataset using Big Data Analytics," Proc of IEEE 2017 International Conference on Infocom Technologies and Unmanned Systems (ICTUS'2017), Dubai, United Arab Emirates(UAE), December 2017. pp 580- 585.

[26] A. Rishika Reddy and P. Suresh Kumar, "Predictive Big Data Analytics in Healthcare," Proc of IEEE 2016 Second International Conference on Computational Intelligence & Communication Technology (CICT), Ghaziabad, 2016, pp. 623-62.