The Practicality of using Virtual Machine Introspection Technique with Machine Learning Algorithms for the Detection of Intrusions in Cloud

A Alfred Raja Melvin[1], Dr. G Jaspher W Kathrine[2], Dr. J Immanuel Johnraja[3]
{alfredraja@karunya.edu.in[1], kathrine@karunya.edu[2],
immanueljohnraja@karunya.edu[3]}
Ph.D Scholar, Dept. of CSE, KITS[1], Assistant Professor,Dept. of CSE, KITS[2],
Associate Professor, Dept. of CSE, KITS[3]

Abstract. This paper presents a novel pattern generation algorithm for the implementation of Virtual Machine Introspection (VMI) based Intrusion Detection System (IDS) for Cloud Computing. The method uses Drakvuf VMI technique for gathering the behavioral characteristics of malware and benign samples. The behavioral characteristics data are then fed to the proposed algorithm for the generation of patterns in-order to generate the dataset. The algorithm includes the generation of frequency distribution of each system calls, hash value based on SHA256 algorithm for the list of file names, hash value based on SHA256 algorithm for the list of process names. Finally, the generated dataset is evaluated using Machine Learning (ML) algorithms with 10-Fold cross validation. It is found that J48 (C4.5) tree classification algorithm performed well with high detection accuracy compared to other ML algorithms. The detection accuracy is 99.1379% for dataset size of 232 instances. As the number of instances in the dataset was increased, the detection accuracy has improved to the maximum of 100% for the dataset size of 273 instances.

Keywords: Virtual Machine Introspection, Virtual Machine Monitor, Intrusion Detection System, Malware, Machine Learning, Cloud Computing.

1 Introduction

Due to the advancement in cloud computing, many enterprises are now relying on cloud computing to process, store and access their data and services. The companies are investing in cloud computing since they only have to pay for the services they use. Also, the services are readily, swiftly available and become accessible from anywhere in the globe. One primary benefit of cloud is its flexibility and scalability i.e., it can be easily scale in or out, scale up or down.

ICASISET 2020, May 16-17, Chennai, India

DOI 10.4108/eai.16-5-2020.2303939

The definition of cloud computing as per Microsoft is “It is the provisioning of computing services like servers, storages, databases, networking, software over the Internet”. In reality, it may be the provisioning of anything over Internet. Typically the deployment model for cloud may be either IaaS or PaaS or SaaS.

The Virtualization technology is the foundational and fundamental for the building up of Cloud Services. In Virtualization technology, the Hypervisor or Virtual Machine Monitor (VMM) is a system software component responsible for creating and managing the Virtual Machines (VM) and its resources like virtual CPU, virtual storage, virtual memory, virtual networks, etc, for the provisioning of services over Cloud. The virtual machines running guest OS (32 bit or 64 bit) can be of any OS flavour like Microsoft Windows XP/7/8/10, Linux OS, BSD OS etc.,. Popular examples of Hypervisor are VMware, VirtualBox, KVM, Xen.

Even though Cloud Computing provides a lot of benefits, it encounters serious security threats. According to the Cisco Annual Security Report 2018 [1], computer attackers are abusing the cloud computing services and its resources for malicious use. A user may register and use the cloud computing resources for launching traditional attacks like DoS, DDoS, Phishing etc., In addition to these, an attacker may launch cloud specific attacks like VM side channel attacks, VM Escape attack VMDoS attack, etc.,

According to Symantec Threat Report (February 2019) [2], there are around 70 million records stolen due to poorly configured S3 buckets at Amazon cloud platform. In addition to this, hardware chip vulnerabilities like Meltdown, Spectre and Foreshadow allow an attacker to get unauthorized access to memory of other cloud instance since they share pools of memory. European Network and Information Security Agency (ENISA) reported that Dropbox was attacked by Distributed Denial of Service attack during Jan 2013 and experienced loss of service for about 15 hours [3]. Security researchers has identified exploit on Amazon cloud platform in their Elastic Search ver 1.1x [4].

Since many threats, vulnerabilities and attacks are possible in cloud environment and its resources, protection to cloud computing components is very essential. Different security solutions are available to ensure protection for traditional computing and network environment. They are Intrusion Detection System, Intrusion Detection and Prevention System, Firewall, Antivirus, and by implementing strong security policies.

IDS solution is very promising and used by many enterprises. However traditional IDS cannot detect cloud specific malicious activities and attacks. Therefore, an implementation of Cloud specific IDS is essential. A Cloud specific IDS based on VMI technique is possible since it provides monitoring of process activities inside the VM from the hypervisor. Security researchers have used VMI technique to study and understand the behaviour of malware samples since it provides features like Isolation, Interception, Live Environment, Memory Snapshot, Introspection, and Stealthy Environment. Cloud administrators can develop and Implement Security Solutions using VMI, without affecting the privacy concerns of the customers. Therefore, VMI based IDS can be designed to detect Intrusions at Cloud Computing.

Many frameworks for VMI are available of which LibVMI is popular and open

source. Drakvuf tool developed by Lengel [5] is based on LibVMI which was implemented over Xen hypervisor. So far, Drakvuf is used only for dynamic malware analysis.

Problem Statement

- The complexity of the existing methods for the implementation of IDS for securing VMs in cloud computing is high.
- There is no standard dataset available based on the behavioural characteristics of malware through virtual machine introspected data.
- The introspected data (log) of Drakvuf needs lot of manual processing in-order to conclude whether the presence of Intrusion is present or not.

This paper presents the study and possibility of implementing VMI based IDS using Drakvuf with Machine Learning algorithms. The following steps are followed to carry out the experiment. First, the Installation and Configuration of analysis environment using Xen virtualization with Drakvuf was done. Secondly, malware and benign samples are submitted to analysis environment and its behavioural characteristics and events are recorded. Thirdly, a proposed feature generation procedure was followed to extract the essential details from the recorded events in-order to generate the dataset with labels. Fourthly and finally, various ML algorithms are used to evaluate the detection accuracy of Intrusions based on 10-fold cross-validation tests and the results are discussed.

The remaining section of this paper are arranged as follows, section-II depicts the existing work, section-III provide details about the proposed IDS architecture, section-IV present details about the System Implementation and Evaluation Results, and section-V discuss about the conclusion and future work.

2 Existing Work

In recent years VMI has emerged as an application for the development of IDS and for dynamic malware analysis due to its ability to gather digital artefacts of virtual machine environment.

Rajendra Patil et al [6] has proposed an in-VM Agent based malware detection system for virtual machine instances in cloud computing. The agent periodically checks for newly launched executables and verify the maliciousness by comparing its signature with the known malware signature database. If the executable signature is unknown, it selects the optimal features using bat algorithm of the binary executable file and send those details to the hypervisor. The hypervisor runs a anomaly based detection system which checks the received features (profile) and decides whether it is malware or not by applying Random Forest classifier. The newly added profile will be used later for the detection of similar malware across all VMs. Even though the system is scalable (able to detect

malware across multiple VMs), recent intelligent malware may not exhibit its execution due to the presence of in-VM agent which poses a challenge.

Bhavesh Borisaniya et al [7] proposed a security framework based on VMI for cloud. Their implementation is based on Nitro tool and supports multi-threaded analysis which has the ability to monitor and detect malicious activities across Virtual Machines hosted on multiple cloud servers.

Ajay Kumara et al [8] proposed an Intelligent Cross View Analyzer based on VMI, Memory Forensics Analysis (MFA) and ML algorithms. They used an in-guest agent inside VM and VMI to get details of the processes. Also they used ML algorithms on the mined information of binary executables gathered using Memory Forensics Analysis. Even though their detection accuracy is good; the complexity of their implementation is very high.

Preeti Mishra et al [9] proposed a security framework based on NIDS and VMI based IDS. According to them NIDS will act as the first layer of protection and then a second level of defence implemented at virtualization layer.

Michael et al [10] proposed a anomaly based malware detection system at hypervisor. They utilized features gathered from system and network level of cloud node. One class Support Vector Machine (SVM) is used in-order to differentiate between malware and benign. Even though the system may detect new type of attacks and malware, the system's detection accuracy is only 90% and it is not scalable.

Mishra et al [11] proposed a system call analysis approach named Malicious System Call Sequence Detection (MSCSD) to detect malware at VM. They used C4.5 ML algorithm to train the classifier to differentiate between malware and benign samples. Even though the system is scalable, the system can be compromised since its implemented inside the VM.

Aristide et al [12] proposed a system named AccessMiner at hypervisor for the detection of malware based on anomaly detection method. The anomaly system is built based on the interaction patterns between the benign programs and operating system during the training phase. Thereafter any application interaction pattern which does not match with the existing are considered as malware. Even though the system may detect zero day malware, their implementation promises only 90% of detection accuracy only.

Marnerides et al [13] proposed malware detection at hypervisor based on the information collected from system and network level from VM. They applied a Ensemble Emprical Mode Decomposition technique to detect the malware. The detection accuracy is around 90% only.

Lengyel et al has developed Drakvuf which is a binary malware analysis engine. It is a VMI based technique, works on the principle of kernel debugging. So far, it’s widely been used in the field of malware analysis.

The hypothesis to carry out this research is to find out whether the Introspected data of Drakvuf VMI technique is really useful for the development of IDS or not. The next section describes the architecture and working of Drakvuf.

3 Proposed IDS Architecture

An IDS consists of three components namely Data source, Sensor and Decision Engine. The Data source depicts the source of data which is applied to train the system. The Sensor component depicts the live monitoring environment. The Decision Engine classifies the data, take decision and raise alarm during the case of intrusions. The architecture of proposed IDS is shown in figure 1. First, the raw data source comes from VMI component. The VM running guest OS provides a real environment for the sample execution. During malware execution, the system traces all system calls (APIs), registry key changes, File names accessed, created, modified, processes impacted. The system call features are extracted using the proposed algorithm to generate the proposed dataset. This dataset is further used to train the model.

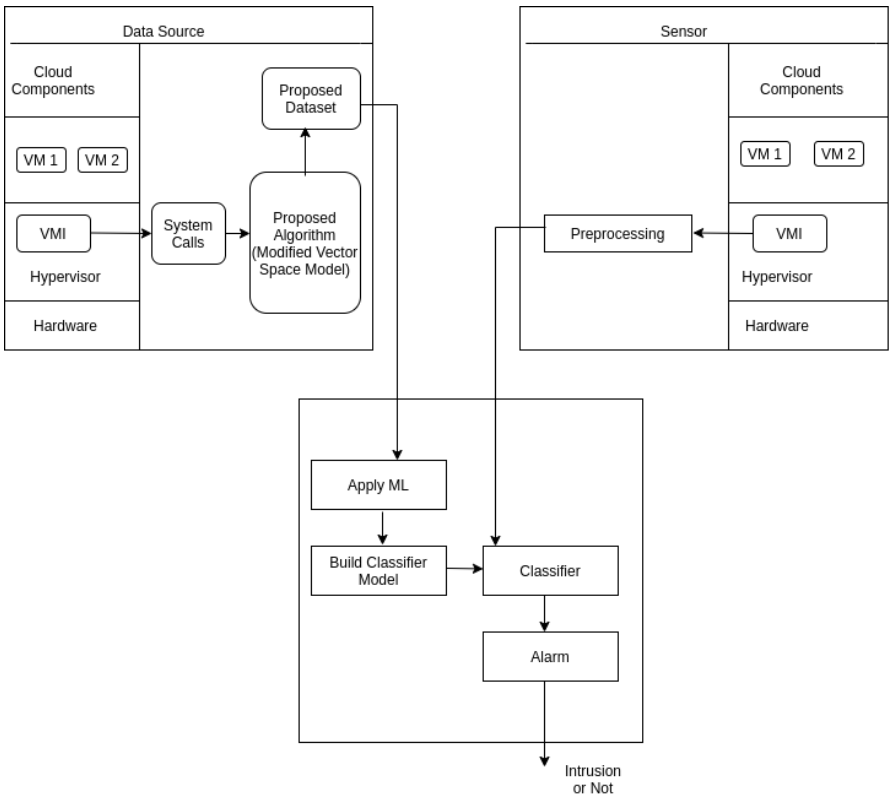


Fig. 1. Architecture of Proposed IDS

The live monitoring environment inside hypervisor is provided by VMI. After pre-processing, the live monitored details are supplied to detection engine. The decision engine makes use of machine learning algorithms to detect the presence of intrusion. An alarm will be raised if intrusion is detected.

4 Experimental Setup and Results

The workflow diagram of our experimental setup is shown in figure 2.

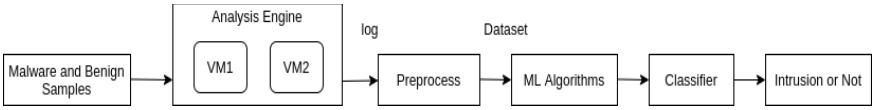


Fig. 2. Workflow of the Experimental Setup

The first step in the experiment is gathering malware samples. Malware samples are collected from theZoo[14] project and Vxheaven [15] archives. Majority of malware collected are viruses and remaining are Trojan horses. All malware collected are based on windows platform, since majority of the malware are written targeting windows platform[17] [18]. Also majority of PC users are using only Microsoft windows platform. In addition to malware samples, regular, free and open source software, in-built windows software is also used. The goal is to train the system to understand the difference between malware and benign sample. The number of malware and benign samples used in the experiment are listed in table 1.

Table 1. Malware and Benign Samples used in the Experiment

#	**Samples**	**Type**	**Count**
1	Viruses	Malware	175
2	Trojans	Malware	50
3	Accessories	Benign	30
4	System Tools	Benign	18
Total # of Samples			273

4.1 Analysis Engine (AE)

The objective of AE is to gather the behavioural characteristics of malware samples. The input for AE is the list of malware samples and benign software. The output of AE is the behavioural characteristics of malware sample and benign software respectively in the form of log file. The Dynamic Malware Analysis Engine is implemented using Xen virtualization technology, LibVMI framework and Drakvuf tool. The system configuration used for AE is Dell Optiplex

workstation with intel core i7 CPU with clock speed of 3.60 GHZ, Hard Disk capacity of 1TB and memory of size 8GB running Ubuntu 16.04 Linux OS as the host machine. The host machine is configured to create and run two virtual machines parallel in-order to execute the malware/benign samples. The VM is configured to run windows 7 OS (64 bit) as guest OS with memory of size 1GB, Hard Disk capacity of 30GB and one dedicated CPU. The number of VMs can be increased based on the limitation of host machine capacity. A simplified shell script was developed in-order to automate the analysis process.

The sample (malware/benign) inside the VM was initiated (started executing) from the hypervisor using Drakvuf based on process injection technique. Immediately after the process was injected, the system traces the system call execution, file access details and processes spawned details for a duration of 60 seconds. The duration is set to 60 sec as mentioned by the author which is enough to gather the behavioural characteristics of any malware. Finally, for each malware or benign sample the system generates a log file which will be further processed for dataset generation.

4.2 Pre-processing

The objective of pre-processing stage is to generate Patterns from the Drakvuf log files in-order to accurately differentiate between malware and benign process. The patterns generated are used to build the Dataset. The Dataset is later used for evaluations.

A novel pattern generation algorithm (algorithm1) was proposed to build the Dataset. In order to generate patterns for the classification of malware and benign samples, three main features are considered. They are API function names (System calls), File names and Process names. The steps involved in the algorithm are as follows. First, API names or System call names of around 227 are gathered from all process logs and are used in the dataset generation process. The reason for including system calls is because its execution pattern helps us to understand the behavioural characteristics of process. In windows OS, the API names starts with Nt are all system calls. The top ten most frequently used system calls and its description is shown in table 2. Second, File names created, deleted, modified or accessed during the process execution with its full pathname are taken for the analysis. Third, Process names are the list of processes spawned during the sample execution is also considered for analysis.

Table 2. Top Ten Windows API Function Names

S.No.	API Function Name	Description
1	NtOpenFile	Windows API Function to open a file or device or directory or volume that exists.
2	NtCreateFile	Windows API Function to create file, directory or opens existing file, device,

		directory or volume.
3	NtWriteFile	Windows API routine to write data to an open file.
4	NtOpenKey	Windows API routine to open an existing registry key.
5	NtCreateKey	Windows API routine to create a new registry key or to open an existing key.
6	NtQueryKey	Windows API routine to provide information about the class of a registry key.
7	NtOpenProcess	Windows API routine to open a handle to a process object and sets the access rights to this object.
8	NtOpenThread	Windows API function to open a handle to a thread object with the access specified.
9	NtQueryInformationProcess	Windows API function used to retrieve the information about the specified process.
10	NtQueryInformationThread	Windows API function used to retrieve information about the specified thread.

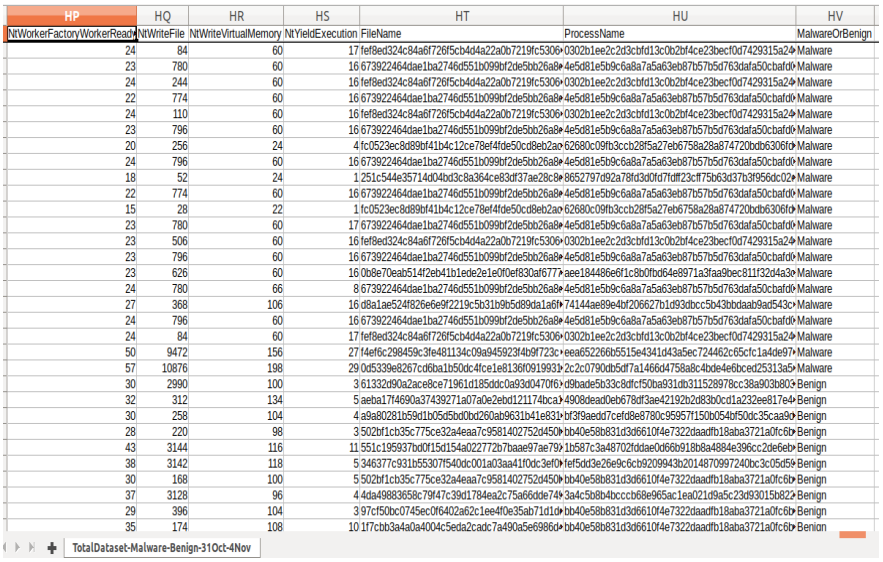
The basic principle behind the pattern generation algorithm for system calls is frequency distribution (i.e., the number of occurrences of a particular system call) from the log file. Next, the list of pathnames (filename) is considered as a string of characters which are supplied to SHA256 algorithm for the generation of 256-bit (32 byte) hash value. Lastly, the lists of processes spawned are considered as a string of characters which are supplied to SHA256 algorithm for the generation of 256-bit (32 byte) hash value. The proposed algorithm to build the dataset is shown below in algorithm1.

Algorithm1: Generate Dataset
Input: Sample Set (S_s), System Call List (S_l)
Output: Dataset, D_s

```
For i in Sample Set (Ss)
        L ← Log file of i
        M ← Metadata of i
        For j in System Call List (Sl)
                C ← Append Count of j in L
        End
        Fs ← List of file names in M
        Ps ← List of process names in M
        Fv ← SHA256 of Fs
        Pv ← SHA256 of Ps
        Append Label, C, Fv, Pv → Ds
End
```

Finally, the frequency values of each system calls (totally 227), SHA256 hash value for the list of file names, SHA256 hash value for the list of process names are used to generate the Dataset. This Dataset generated contains a total of 230

features including the label for classification. Each record in the dataset is labelled as either malware or benign. This dataset is used in-order to train the classifier. This dataset is made available public through github[16]. The snapshot of the dataset is shown in figure 3.



HP	HQ	HR	HS	HT	HU	HV
NtWorkerFactoryWorkerReady	NtWriteFile	NtWriteVirtualMemory	NtYieldExecution	FileName	ProcessName	MalwareOrBenign
24	84	60	17	fef8ed324c84a6f726f5cb4d4a22a0b7219fc5306	0302b1ee2c2d3cbfd13c0b2bf4ce23becf0d7429315a24	Malware
23	780	60	16	673922464dae1ba2746d551b099bf2de5bb26a8e	4e5d81e5b9c6a8a7a5a63eb87b57b5d763dafa50cbafd6	Malware
24	244	60	16	fef8ed324c84a6f726f5cb4d4a22a0b7219fc5306	0302b1ee2c2d3cbfd13c0b2bf4ce23becf0d7429315a24	Malware
22	774	60	16	673922464dae1ba2746d551b099bf2de5bb26a8e	4e5d81e5b9c6a8a7a5a63eb87b57b5d763dafa50cbafd6	Malware
24	110	60	16	fef8ed324c84a6f726f5cb4d4a22a0b7219fc5306	0302b1ee2c2d3cbfd13c0b2bf4ce23becf0d7429315a24	Malware
23	796	60	16	673922464dae1ba2746d551b099bf2de5bb26a8e	4e5d81e5b9c6a8a7a5a63eb87b57b5d763dafa50cbafd6	Malware
20	256	24	4	fc0523ec8d89bf41b4c12ce78ef4fde50cd8eb2ae	62680c09fb3ccb28f5a27eb6758a28a874720bdb6306f0	Malware
24	796	60	16	673922464dae1ba2746d551b099bf2de5bb26a8e	4e5d81e5b9c6a8a7a5a63eb87b57b5d763dafa50cbafd6	Malware
18	52	24	1	251c544e35714d04bd3c8a364ce83df37ae28c8e	8652797d92a78fd3d0fd7dff23cff75b63d37b3f956dc02	Malware
22	774	60	16	673922464dae1ba2746d551b099bf2de5bb26a8e	4e5d81e5b9c6a8a7a5a63eb87b57b5d763dafa50cbafd6	Malware
15	28	22	1	fc0523ec8d89bf41b4c12ce78ef4fde50cd8eb2ae	62680c09fb3ccb28f5a27eb6758a28a874720bdb6306f0	Malware
23	780	60	17	673922464dae1ba2746d551b099bf2de5bb26a8e	4e5d81e5b9c6a8a7a5a63eb87b57b5d763dafa50cbafd6	Malware
23	506	60	16	fef8ed324c84a6f726f5cb4d4a22a0b7219fc5306	0302b1ee2c2d3cbfd13c0b2bf4ce23becf0d7429315a24	Malware
23	796	60	16	673922464dae1ba2746d551b099bf2de5bb26a8e	4e5d81e5b9c6a8a7a5a63eb87b57b5d763dafa50cbafd6	Malware
23	626	60	16	0b8e70eab514f2eb41b1ede2e1e0f0ef830af677	aee184486e6f1c8b0fbd64e8971a3faa9bec811f32d4a3	Malware
24	780	66	8	673922464dae1ba2746d551b099bf2de5bb26a8e	4e5d81e5b9c6a8a7a5a63eb87b57b5d763dafa50cbafd6	Malware
27	368	106	16	d8a1ae524f826e6e9f2219c5b31b9b5d89da1a6f	74144ae89e4bf206627b1d93dbcc5b43bbdaab9ad543c	Malware
24	796	60	16	673922464dae1ba2746d551b099bf2de5bb26a8e	4e5d81e5b9c6a8a7a5a63eb87b57b5d763dafa50cbafd6	Malware
24	84	60	17	fef8ed324c84a6f726f5cb4d4a22a0b7219fc5306	0302b1ee2c2d3cbfd13c0b2bf4ce23becf0d7429315a24	Malware
50	9472	156	27	f4ef6c298459c3fe481134c09a945923f4b9f723c	eea652266b5515e4341d43a5ec724462c65cfc1a4de97	Malware
57	10876	198	29	0d5339e8267cd6ba1b50dc4fce1e8136f091993b	2c2c0790db5df7a1466d4758a8c4bde4e6bced25313a5	Malware
30	2990	100	3	61332d90a2ace8ce71961d185ddc0a93d0470f6d	d9bade5b33c8dfcf50ba931db3115289878cc38a903b803	Benign
32	312	134	5	aeba17f4690a37439271a07a0e2ebd121174bca	4908dead0eb678df3ae42192b2d83b0cd1a232ee817e4	Benign
30	258	104	4	a9a80281b59d1b05d5bd0bd260ab9631b41e831e	bf3f9aedd7cefd8e8780c95957f150b054bf50dc35caa9d	Benign
28	220	98	3	502bf1cb35c775ce32a4eaa7c9581402752d450b	bb40e58b831d3d6610f4e7322daadfb18aba3721a0fc6b	Benign
43	3144	116	11	551c195937bd0f15d154a022772b7baae97ae799	1b587c3a48702fddae0d66b918b8a4884e396cc2de6eb	Benign
38	3142	118	5	346377c931b55307f540dc001a03aa41f0dc3ef0	fef5dd3e26e9c6cb9209943b2014870997240bc3c05d59	Benign
30	168	100	5	502bf1cb35c775ce32a4eaa7c9581402752d450b	bb40e58b831d3d6610f4e7322daadfb18aba3721a0fc6b	Benign
37	3128	96	4	4da49883658c79f47c39d1784ea2c75a66dde749	3a4c5b8b4bcccb68e965ac1ea021d9a5c23d93015b822	Benign
29	396	104	3	97cf50bc0745ec0f6402a62c1ee4f0e35ab71d1d	bb40e58b831d3d6610f4e7322daadfb18aba3721a0fc6b	Benign
35	174	108	10	1f7cbb3a4a0a4004c5eda2cadc7a490a5e6986d4	bb40e58b831d3d6610f4e7322daadfb18aba3721a0fc6b	Benign

TotalDataset-Malware-Benign-31Oct-4Nov

Fig. 3. Snapshot of the Generated Dataset

4.3 Classification and Results

The objective of this component is to build a classifier based on suitable machine learning algorithms, and evaluate the detection accuracy. The input for this stage is the generated Dataset and output is the evaluation results.

The popular and open source machine learning tool, WEKA is used to evaluate the generated dataset. The screenshot of WEKA tool after loading the dataset is shown in figure 4. The dataset contains a total of 273 Instances (Records), with 225 instances of malware and 48 instances of benign samples. All experiments are carried out with 10-fold cross validation. The popular classification algorithms namely J48 (C4.5), Random Forest, JRip and NaiveBayes are evaluated on the dataset. J48 (C4.5) and Random Forest are based on tree classifier, JRip is based on rule classifier and Naïve Bayes is based on bayes theorem.

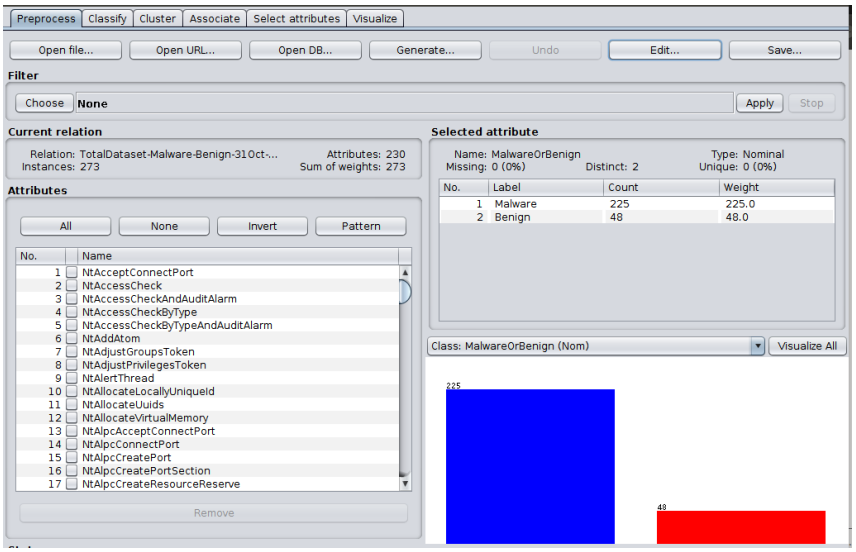


Fig. 4. Dataset loaded in WEKA tool

Classification using C4.5 algorithm: It is a Machine Learning algorithm to generate a decision tree which is used for classification and it is often referred to as statistical classifier. A binary classification using C4.5 (J48) algorithm is applied using WEKA tool. The classes in the dataset are Malware (Intrusion) and Benign (Normal). The decision tree (pruned) generated by this algorithm is shown in figure 5. The size of the generated tree is 7 with four numbers of leaves. From the pruned tree, it is clear that the system calls – NtcreateToken, NtReleaseWorkerFactoryWorker, NtAllocateLocallyUniqueid are significant in making decision. The time taken to build the model is 0.18 seconds. Referring to table 3, 273 out of total 273 instances are correctly classified giving the intrusion detection accuracy of 100% with zero false positive rates.

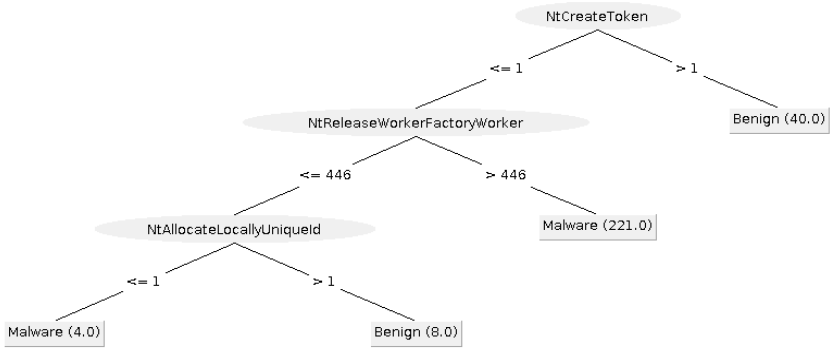


Fig. 5. Pruned Decision Tree – C4.5 Classifier

Random Forest: It is a Machine Learning algorithm which operates by constructing multitude of decision trees. It can be applied for solving classification and regression problems. The time taken to build the classifier for our dataset by this algorithm is 0.36 seconds. The evaluation result of this algorithm is shown in table 3. 271 out of 273 instances are correctly classified giving a detection accuracy of 99.2674% and false positive rate of 0.034.

JRip. It is a Machine Learning algorithm which implements a propositional rule learner called Repeated Incremental Pruning to Produce Error Reduction (RIPPER). The time taken to build the classifier for our dataset by this algorithm is 0.17 seconds. The evaluation result of this algorithm is shown in table 3. 272 out of 273 instances are correctly classified giving a detection accuracy of 99.6337% and false positive rate of 0.017.

NaiveBayes. It is a Machine Learning algorithm based on Baye's theorem. It assumes that the presence of particular feature in a class is unrelated to another feature. The time taken to build the classifier for our dataset by this algorithm is 0.1 seconds. The evaluation result of this algorithm is shown in table 3. 21 instances are incorrectly classified and 252 out of 273 instances are correctly classified giving a detection accuracy of 92.3077% and false positive rate of 0.049.

The evaluation results during the experiment is shown in table 3,

Table 3. Evaluation Summary for Generated Dataset (273 Instances)

ML Algorithms **Parameters**	**J48**	**Random Forest**	**JRip**	**NaiveBayes**
Correctly classified	273	271	272	252
Incorrectly Classified	0	2	1	21
TP Rate	1.000	0.993	0.996	0.923
FP Rate	0.000	0.034	0.017	0.049
F Measure (average)	1.000	0.993	0.996	0.927
Detection Accuracy	100%	99.2674%	99.6337%	92.3077%

In-order to evaluate the performance of the proposed system, the following metrics are considered,

Accuracy: It is the measure by which the algorithm can classify correctly between the malware and benign sample, i.e., the ability to differentiate between positive and negative instances. The comparison graph of detection accuracy on ML algorithms is shown in figure 5. The detection accuracy of C4.5 (J48) is 100%, JRip is 99.63%, Random Forrest is 99.27% and NaiveBayes is 92.31%. Since the detection accuracy of C4.5 algorithm is higher, we can claim that it is the best classifier for the proposed system. However, the dataset is not balanced (refer to table 1). Therefore, F1 Score (F measure) metric is also considered for analysis. Accuracy is calculated by the formula (Equation 1),

$$\frac{TP + TN}{TP + TN + FP + FN} \qquad \text{Eq. 1}$$

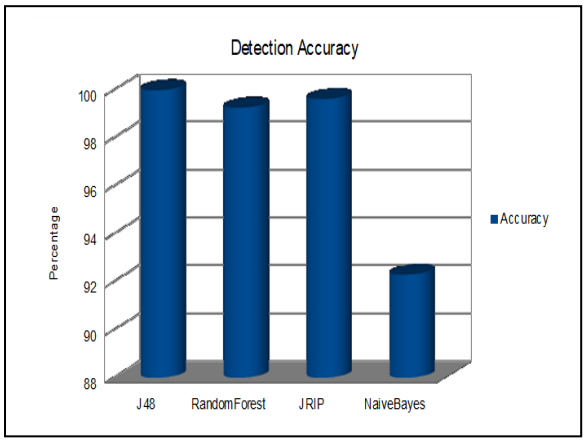


Fig. 5. Detection Accuracy

TP refers to True Positive, TN refers to True Negative, FP refers to False Positive and FN refers to False Negative in the formula.

F Measure (F1 Score): It is the harmonic mean of precision (p) and recall (r). Precision measures the proportion of the detected intrusion that are actually intrusion. Recall measures the proportion of intrusions that are correctly identified. The average (between intrusion and normal) F Measure value for J48, Random Forrest, JRip and NaiveBayes are 1.000, 0.993, 0.996 and 0.927 respectively. The F measure value ranges between 0 to 1. Since the F measure value for J48 is the highest, we conclude that the model built using J48 classifier performs well. The comparison graph on F measure values for ML algorithms is shown in figure 6. F1 score is given by the formula (Equation 2),

$$\frac{2 \cdot pxr}{p + r} \qquad \text{Eq. 2}$$

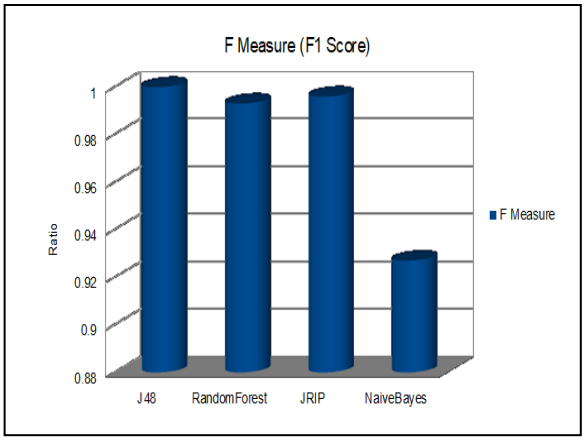


Fig. 6. F1 Score

False Positive (FP) Rate: It is the measure by which the system raises an alarm as intrusion (malicious) but it is not an intrusion. The higher the false positive rate, the higher will be false alarm. For any efficient IDS system, false alarm must be as less as possible. The FP rate values on J48, Random Forest, JRip and Naive Bayes algorithms on proposed system are 0.000, 0.034, 0.017 and 0.049 respectively. Since J48 has less false positive compared to others, we conclude that J48 is best suitable for the proposed system. The comparison graph on FP rate values for ML algorithms is shown in figure 7. It is calculated by the formula (Equation 3),

$$\frac{FP}{FP + TN} \qquad \text{Eq. 3}$$

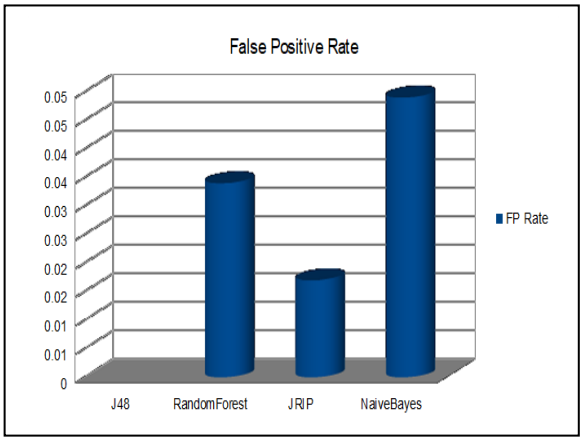


Fig. 7. False Positive Rate

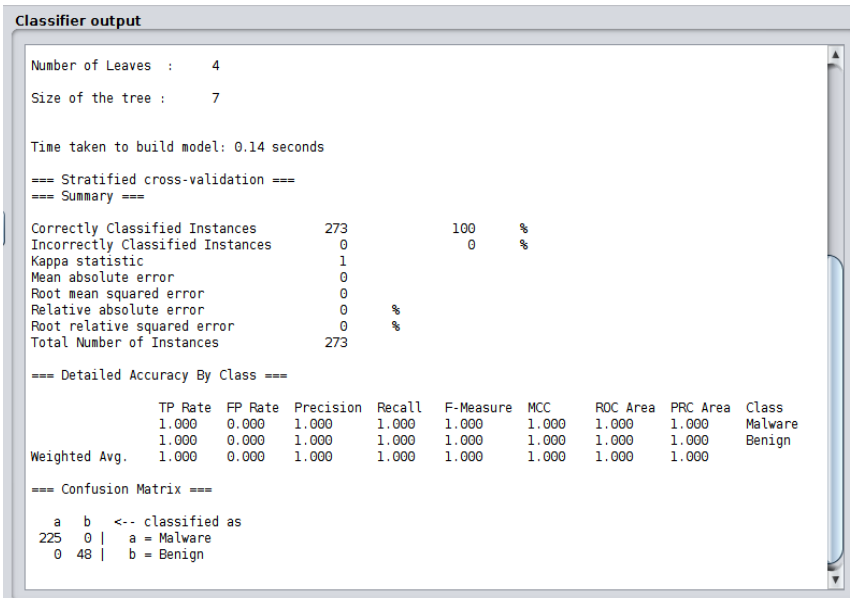


Fig. 8. Evaluation Results of J48 Tree Classifier

From table3 and figures 5 to 7, it is clear that J48 (C4.5) tree classifier is best suitable for the proposed system, since it has the highest detection accuracy and least False Positive rate. The evaluation result of J48 Tree Classifier is shown in figure 8.

Therefore, our proposed pattern generation algorithm for Drakvuf Virtual Machine Introspected data with J48 (C4.5) tree classifier ML algorithm is suitable to detect the Intrusions at Virtual Machines of Cloud Computing.

5 Conclusion and Future Work

This paper presented the possibility of implementing VMI based IDS with Machine Learning algorithms. We explained the proposed architecture of IDS and presented the steps in creating the dataset from VMI introspected system call data.

Based on the Experiments conducted and Evaluation results, it is concluded that the proposed IDS with C4.5 (J48) Machine Learning algorithm performs better compared to Random Forrest, RIPPER and NaiveBayes ML algorithms in-terms of detection accuracy and F Measure. The detection accuracy of proposed IDS with J48 is 0.73% better than Random Forrest, 0.37% better than RIPPER and 7.7% better than NaiveBayes algorithm as shown in table 3. The F Measure shows that J48 algorithm score is higher compared to other algorithms as depicted in figure 6. Also, the False Positive Rate in C4.5 algorithm is less compared to other ML algorithms as depicted in figure 7. Therefore, the proposed IDS with C4.5

(J48) ML algorithm is best suitable for implementing Intrusion Detection System for Cloud. The following are the contributions of this research,

i. This research produces a new dataset based on virtual machine Introspected data.

ii. It provides a practical method on implementing VMI based IDS for cloud.

In continuation of this work, the same classifier (J48) will be evaluated with supplied Test Data using the proposed pattern generation algorithm on the Live Virtual Machine Environment. The future work is to develop an Intrusion Detection System (IDS) Application based on Virtual Machine Introspection Data with Machine Learning and proposed pattern generation algorithm.

Acknowledgments

The authors thank the management and the department for providing support and computing facilities (Computing Security Research Lab) to carry out this research.

References

[1] Cisco Annual Security Report 2018. https://www.cisco.com/c/dam/m/hu_hu/campaigns/security-hub/pdf/acr-2018.pdf, 2018.

[2] Symantec Threat Report 2019. https://www.symantec.com/content/dam/symantec/docs/reports/istr-24-2019-en.pdf, 2019.

[3] European Union Agency for Cybersecurity. https://www.enisa.europa.eu/publications/incident-reporting-for-cloud-computing, 2013.

[4] Amazon cloud infested with DDoS botnets. http://www.techwalls.com/amazon-cloud-infested-ddos-botnets/, 2014.

[5] Lengyel, T.K., Maresca, S., Payne, B.D., Webster, G.D., Vogl, S., Kiayias, A.: Scalability, fidelity and stealth in the drakvuf dynamic malware analysis system. In-Proceedings of the 30th Annual Computer Security Applications Conference. ACM; 2014. 386 – 395.

[6] Rajendra Patil, Harsha Dudeja, Chirag Modi.: Designing in-VM-assisted lightweight agent-based malware detection framework for securing virtual machines in cloud computing. International Journal of Information Security. Springer, 2019.

[7] Borisaniya, B., Patel, D.: Towards virtual machine introspection based security framework for cloud. Indian Academy of Sciences. Springer, 2019.

[8] Ajay Kumara, M.A., Jaidhar, C.D.: Leveraging virtual machine introspection with memory forensics to detect and characterize unknown malware using machine learning techniques at hypervisor. Digital Investigation, 2017. 99 - 123.

[9] Preeti Mishra, Emmanuel S. Pilli, Vijay Varadharajan, Udaya Tupakula.: NvCloud-IDS - A Security Architecture to Detect Intrusions at Network and Virtualization Lay-

er in Cloud Environment. International Conference on Advances in Computing. Communications and Informatics (ICACCI), 2016. 56 - 62.
[10] Michael R. Watson, Noor-ul-hassan Shirazi, Angelos K. Marnerides, Andreas Mauthe, David Hutchison.: Malware Detection in Cloud Computing Infrastructures. IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, 2016, 192 - 205.
[11] Preeti Mishra, Emmanuel S. Pilli, Vijay Varadharajan, Udaya Tupakula.: Securing Virtual Machines from Anomalies using Program-Behavior Analysis in Cloud Environment. International Conference on High Performance Computing and Communications, 2016. 991 - 998.
[12] Aristide Fattori, Andrea Lanzi, Davide Balzarotti, Engin Kirda.: Hypervisor-based malware protection with AccessMiner. Computers and Security. 52, 33 - 50 (2015).
[13] Angelos K. Marnerides, Petros Spachos, Periklis Chatzimisios, Andreas U. Mauthe.: Malware Detection in the Cloud under Ensemble Empirical Mode Decomposition. International Conference on Computing, Networking and Communications and Information Security, 2015. 82 - 88.
[14] TheZoo Malware Collection. http://github.com/ytisf/theZoo
[15] Computer virus collection. http://archive.org/details/vxheaven-windows-virus-collection
[16] Dynamic Malware Dataset based on VMI. https://github.com/aarmelvin/dynamic-malware-analysis.
[17] K. Vijayakumar, Chokkalingam Arun, “Integrated cloud-based risk assessment model for continuous integration”,Int. J. Reasoning-based Intelligent Systems”, Vol. 10, Nos. 3/4, 2018.
[18] K. Vijayakumar, S. Suchitra and P. Swathi Shri, “A secured cloud storage auditing with empirical outsourcing of key updates”, Int. J. Reasoning-based Intelligent Systems, Vol. 11, No. 2, 2019.