Windows Malware Detection Based on Cuckoo Sandbox Generated Report Using Machine Learning Algorithm

Shiva Darshan S.L.[1], Ajay Kumara M.A.[2], and Jaidhar C.D.[3]
Department of Information Technology
National Institute of Technology Karnataka
Surathkal, Mangalore, India
it15f02.shivadarshan@nitk.edu.in[1], ajayit13f01@nitk.edu.in[2], jaidharcd@nitk.edu.in[3]

***Abstract*—Malicious software or malware has grown rapidly and many anti-malware defensive solutions have failed to detect the unknown malware since most of them rely on signature-based technique. This technique can detect a malware based on a pre-defined signature, which achieves poor performance when attempting to classify unseen malware with the capability to evade detection using various code obfuscation techniques. This growing evasion capability of new and unknown malwares needs to be countered by analyzing the malware dynamically in a sandbox environment, since the sandbox provides an isolated environment for analyzing the behavior of the malware. In this paper, the malware is executed on to the cuckoo sandbox to obtain its run-time behavior. At the end of the execution, the cuckoo sandbox reports the system calls invoked by the malware during execution. However, this report is in JSON format and has to be converted to MIST format to extract the system calls. The collected system calls are structured in the form of N-Grams, which help to build the classifier by using the Information Gain (IG) as a feature selection technique. A comprehensive experiment was conducted to perceive the best fit classifier among the chosen classifiers, including the Bayesian-Logistic-Regression, SPegasos, IB1, Bagging, Part, and J48 defined within the WEKA tool. From the experimental results, the overall best performance for all the selected top N-Grams such as 200, 400, and 600 goes to SPegasos with the highest accuracy, highest True Positive Rate (TPR), and lowest False Positive Rate (FPR).**

***Keywords*—Sandbox, Malware Detection, Machine Learning, Hypervisor, Virtual machine, N-Gram Feature Extraction.**

I. Introduction

Malware is also known as malicious software. It is a malicious code developed with the intention of damaging the function of a system. Malware has the capacity to disorder the normal operation by infecting the system or network [1]. It enters a system either through multiple media or gets downloaded into the system as a genuine application. Once it gets into the system, it checks for vulnerabilities and infects the system, if the system is highly vulnerable. Generally, antimalware defensive solutions are signature dependent and run inside the host machines. They are inadequate to thwart the emerging advanced malware attacks.

Computerized malware examination frameworks (or sandboxes) [2] [3] are one of the most recent security innovation used to detect malware based on behavior traits. Such frameworks allow an unknown malware to execute in an isolated environment and screen its run-time behavior. Such frameworks have been in use as a major aspect of the manual investigation process for a while; they are progressively utilized as a primary component of the automated malware detection approach. The main upside of the automated malware detection technique is that it is able to recognize the unseen malware on the basis of the observed activities gathered during the execution of the malware. Majority of the sandboxes observe at the system call interface the behavior of a user mode process. System calls are a routine that allow the operating system to interact with the user-level process to perform their desired task. These tasks include reading data from files, delivering packets across the network, and recording of entry from the registry. Looking deeper into the execution of a program, a lot more interesting information can be gathered.

This paper presents a classic approach to the detection of malware by extracting only the system calls (i.e., operation field) from the Malware Instruction Set (MIST) report that were obtained by implementing the MIST conversion process for all those runtime behavioral reports of malware produced by cuckoo sandbox. Further, the extracted system calls are used to generate the sequence of N-Grams of specified length such as N=2, N=3, and N=4, and then, adopt the Information Gain (IG) feature selection method to calculate a score for each N-Gram. Later, the top N-Grams are selected based on the highest IG score. The selected top N-Grams are processed by the classifier for classification.

The rest of the paper is organized as follows. In Section II, we study the background of MIST instruction and its representation. In Section III, we review earlier research to detect malicious executables. In Section IV, we describe our proposed approach. In section V, experimental results are discussed. Finally, conclusion is drawn in the Section VI.

II. BACKGROUND

The prime task of the malware detection system is to identify known as well as unknown malware and defend the integrity of the system, while performing its function. The analysis of the malware can be performed in two ways i.e.,

code analysis, and behavior analysis. The Code analysis is generally achieved in a static way by obtaining a complete overview of the software. A major limitation of the code analysis technique is that it is often clogged by evasion techniques such as binary packers, polymorphism, and anti-debug techniques. In behavior analysis, the malware behavior is monitored, while it is running on a host system. Behavior-based malware analysis is an efficient way of observing the actions of the malware, while several existing monitoring tools provide the behavioral report [3]. Generally, behavioral-based malware analysis tools execute a malware sample in an isolated environment to obtain accurate system level behavior by monitoring and recording the system calls invoked by the malware. A summarized observed behavior of the malware sample is tabulated in the analysis report. Monitoring suites such as Anubis and CWSandbox produce the behavior report in textual or XML-based format that provide system-level behavior of the malware, that includes system calls details. A human-analyst can easily analyze textual or XML-based formats as they are unsuitable for further automatic analysis due to a negative impact on the runtime of the analysis. XML representations are inappropriate for finding generic behavioral patterns. Unlike XML, textual representations are tough due to aggregation and even increase the size of the report. In contrast to textual and XML-based format, a MIST is used to record all system level behavior in which the system call arguments are organized in different levels of blocks (Fig. 1).

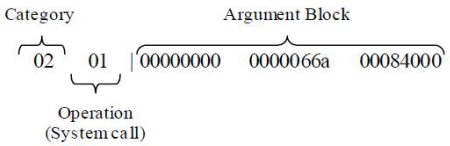


Fig. 1: MIST representation of system call.

The first field category denotes the type of system calls and the second field operation represents a particular system call. In each MIST instruction, the type of the argument block and its size depends on the particular system call. The MIST representation is an optimized form for an effective and efficient way of analyzing the malware behavior using machine learning algorithms [4].

III. RELATED WORK

There have been several dynamic malware sandbox approaches proposed in literature that perform dynamic malware analysis using sandbox technology. Willems et al. [5] developed an open source tool called CWSandbox that allows a malware sample to execute either in a native environment or in a virtual Windows environment. Monitoring of the API calls is accomplished by the hook functions of analysis component. The DRAKVUF [6] is another dynamic malware analysis system that performs insight trace analysis of execution of malware, including modern stealthy kernel rootkit by intercepting the kernel heap allocation of the targeted system. In addition, DRAKVUF efficiently addresses the challenges in the detecting the system call interception by other sandbox systems [5]. On the other hand, virtualization-based sandbox techniques [2] [7] play a vital role by examining the manipulated structure of the operating system that is caused by the types and behavior of new variants of malware.

Cuckoo [3] is another malware analysis system, which provides a detailed behavior report of a Windows executable file, when executed inside an isolated environment. Cuckoo can analyze many different malicious files (executables, document exploits, etc.) and malicious web-sites in a virtualized environment. Cuckoo is able to trace the API calls and general behavior of the input file and can easily integrate within the existing framework. The current development of the sandbox based system [8] [9] is sufficient in providing behavior activity of input an executable file in the form of a behavioral report. However, an accurate examination of the malware based on the sandbox generated report involves extensive manual analysis. In addition, the sandbox also provides a report for benign executables files on the monitored machine. In such cases, precisely detecting actual malware activities from other benign executable applications is a challenging task. The sandbox report is available in an unstructured form to precisely extract actual semantic information (e.g, system call). Authors Rick et al. [4] made an attempt to form an effective detection of malware based on the invoked system call sequence. The collected system call sequence structured in the form of N-Grams and N-Gram feature extraction technique is widely used for different input sources [10] [11] [12]. In another work, Tesauro et al. [13] applied the idea of N-Grams as features for malware detection. The N-Grams were selected from most frequent classes in malware and benign files. The N-Grams outperform when the experiment is carried with a larger feature set. Recent reports have shown that feature selection based on the IG has produced the best results in classifying malicious executables files from benign executable files [10].

Machine learning algorithms are witnessed as a promising technique to perform an accurate detection of malicious malware from benign executable files. Kolter et al. [14] describe machine learning algorithm to classify the malicious executables that appear in the wild by encoding the N-Grams as features for classification. Automated behavior-based malware analysis framework using machine learning technique was proposed [15] that convert the report generated by the sandbox into MIST format to identify the unknown malware with similar behavior.

In our work, we have used the cuckoo sandbox to gather the system-level behavior of the executable files. The system calls' sequence, triggered by the executable files (processes), are extracted from the cuckoo sandbox generated report. IG feature selection technique is employed to choose the best features to construct the Final Feature Vector (FFV). Machine learning algorithm is employed to classify the malware executable files from benign executables files based on the FFV.

IV. PROPOSED WORK

Our proposed work distinguishes the malware files from benign files on the basis of system calls' sequence is structured using a heuristic method called N-Grams analysis. It adopts the IG technique to compute the IG score for the each N-Gram and extracts the top N-Grams (features) based on the highest IG score in order to prepare a FFV that is needed for classification. Fig.2 depicts an overview architecture of the proposed work.

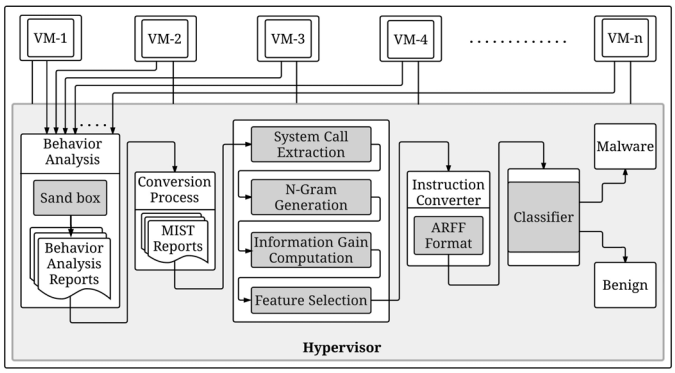


Fig. 2: System Architecture of the proposed work.

A. Behavior analysis

Since, the cuckoo sandbox functions at hypervisor as a separate entity, it examines the behavior of malware which are running on VMs to obtain the behavioral analysis report of running executables in JavaScript Object Notation (JSON) format.

B. Conversion process

The analysis reports obtained in JSON format are pre-processed to obtain the MIST, since it is a preferred format that uses a smaller file size and reduces processing time. Since our approach is specific to observation on monitored system calls, we are concerned with the operation field (system call as shown in Fig. 1) of MIST files to generate N-Grams (4 bytes) files as shown in Fig. 3.

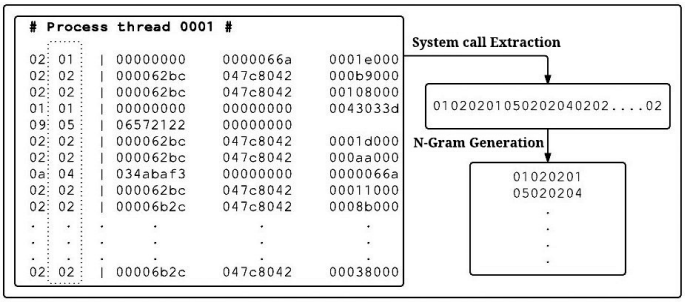
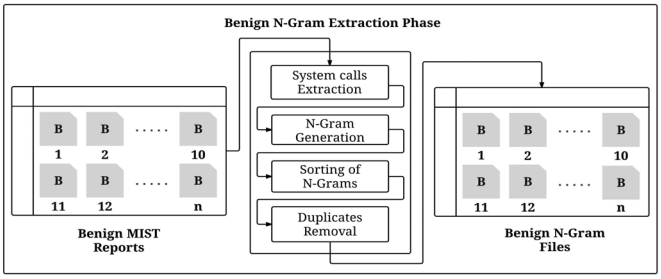


Fig. 3: Snippet of N-Gram extraction using MIST file.

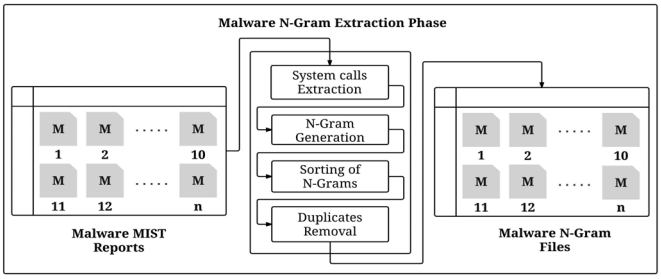
To generate the N-Gram files, we follow the following steps:

- System calls extraction,
- N-Gram generation,
- Sorting of N-Grams, and
- Duplicate removal

In *first step* system call extraction, we select only the operation field, i.e., the system calls of all the benign MIST files (1, 2, ...,10, 11, ... n) and all the malware MIST files (1, 2, ...,10, 11, ... n) as shown in Fig. 4, Since we have the record of all system level behaviors. The extracted operation fields are stored in a text file and grouped in sequence to form N-Grams of variable length, i.e., N=2, N=3, N=4, etc. The lengthier the N-Grams size, better characteristics are represented. A snippet of extraction is as shown in Fig. 3. We have grouped N-Grams of length four bytes, while forming the N-Grams in the *second step* of the generation phase. In the *third step*, the formed N-Grams are sorted in descending order to get the highest order sequence of N-Grams. After the sorting operation in the *fourth step*, the duplicates should be removed, if observed to get unique N-Grams. The unique N-Grams can be employed for better feature selection and also provide better classification.



(a) Steps to generate Benign N-Gram Files.



(b) Steps to generate Malware N-Gram Files.

Fig. 4: System call extraction phase.

The above explanation is prerequisite for the feature selection approach, since it cannot be performed without the N-Gram formation. The formed Benign N-Gram files [B1, B2, B3,. . . ,Bn] and Malware N-Gram files [M1, M2, M3, . . .,Mn] must undergo union operation considering each benign N-Gram files [B1 ∪ B2 ∪ B3 ∪ . . . ∪ Bn] and malware N-Gram files [M1 ∪ M2 ∪ M3 ∪ . . . ∪ Mn]. After the union operation, the benign union N-Gram files and malware union N-Gram files must be sorted in non-increasing order and duplicates must be removed, if observed to achieve unique benign N-Gram files and unique malware N-Gram files. The occurrences of each unique benign N-Gram in the benign N-Gram files are observed and tabulated as N-Gram frequency table for the benign class, and in the same way, the occurrences

of each unique malware N-Gram in the malware N-Gram files are observed and tabulated as N-Gram frequency table for the malware class.

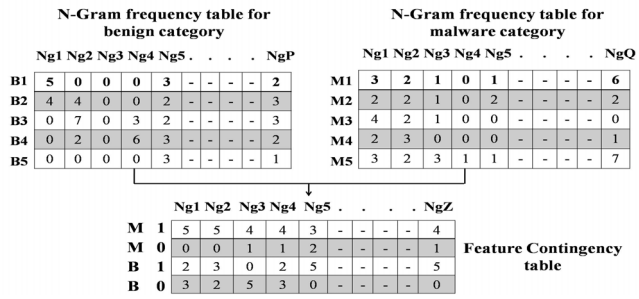


Fig. 5: N-Gram frequency table for benign class and malware class with feature contingency table.

The feature contingency table is then prepared based on the values accommodated in the N-Gram frequency table for benign category and malware category as depicted in Fig. 5. The feature contingency table is used to calculate Information Gain [10]. Information Gain is computed by the following equation,

$$IG(N-Gram) = \sum_{v_{N-Gram} \in \{0,1\}} \sum_{C \in \{C_i\}} P(v_{N-Gram}, C) \log \frac{P(v_{N-Gram}, C)}{P(v_{N-Gram}), P(C)} \quad (1)$$

Where, C is one of the two categories - benign or malware and v_{N-Gram} is the value of N-Gram. v_{N-Gram} = 1 indicates that the N-Gram present either in benign N-Gram files or malware N-Gram files and v_{N-Gram} = 0, otherwise. P(v_{N-Gram}, C) is the proportion of N-Gram files in C in which the N-Gram takes on value v_{N-Gram}. P(v_{N-Gram}) is the proportion of benign N-Gram files or malware N-Gram files in entire training set such that N-Gram takes the value v_{N-Gram} . P(C) is the proportion of data set belonging to category C. The N-Grams are organized in non-increasing order based on the IG score and the topmost L number of N-Grams are extracted as best features for classification purpose.

C. Instruction Converter

The instruction converter converts the extracted features into an ARFF (Attribute-Relation File Format) file. ARFF is an ASCII text file that describes a list of instances sharing a set of attributes. It is an important process because the classifiers of WEKA tool used in our approach works with the ARFF file.

V. EXPERIMENT RESULTS

Our experimental data consists of 3000 benign MIST files and 3100 malware MIST files. The malware MIST files consists of four different families such as Swizzor (1000), Basun (1000), AutoIt (1000), and Kelihos Trojan (100). Among the considered four different malware families the first three were collected from the public source [1] and the remaining 100 malware MIST files were obtained by implementing the MIST conversion process for all those runtime behavioral reports produced by cuckoo sandbox by injecting the Kelihos Trojan. As explained earlier, we extracted N-Grams of different sizes 2bytes, 3bytes and 4bytes to measure which N-Gram size achieves the best detection rate. A separate experiment was conducted for each N-Gram size.The N-Grams are sorted in decreasing order based on the IG score and duplicate N-Gram is removed, if found. The class-wise document frequency for each class was determined for each N-Gram to prepare the contingency table. The IG method is used to calculate a score for each N-Gram and the top K N-Grams are determined based on the highest IG score. Experiment were conducted for different values of K such as 200, 400, and 600. Further, the best features were drawn at each K value for different N-Gram lengths. The best features were pre-processed through the instruction converter to prepare ARFF files for the selected N-Grams. The ARFF files were submitted to the WEKA tool for classification. A wide set of experiments were conducted to determine which classifier achieved best malware detection rate with low False Positive Rate (FPR). We evaluated the performance of several classification algorithms stated in the WEKA tool.

Our objective was to know the best classification algorithm among the several stated in the WEKA tool. From that perspective, we selected six classifiers among the eight different categories mentioned in the WEKA tool. The six classifiers chosen were the Bayesian-Logistic-Regression, SPegasos, IB1, Bagging, Part and J48 classified under Bayes, functions, lazy, meta, rules and trees of WEKA. For evaluation purposes, we measured and tabulated the values of True Positive Rate (TPR), False Positive Rate (FPR), Precision, Recall, F-measure, ROC Area and Accuracy for all the chosen six classifiers as shown in TABLE I and TABLE II.

Two experiments were carried out by us: In the *first experiment*, we considered N-Gram of three bytes in order to select the top N-Grams based on the highest score of IG. The top N-Grams were selected in terms of 200, 400, and 600. From the experimental observation, as shown in Fig. 6, the highest accuracy was 89.77% for 200 N-Grams, 90.03% for 400 N-Grams, and 89.88% for 600 N-Grams yielded by the SPegasos classifier (Fig. 6a). The highest TPR of 0.898 for 200 N-Grams, 0.9 for 400 N-Grams, and 0.899 for 600 N-Grams was produced by the SPegasos classifier (Fig. 6b). The lowest FPR of 0.102 for 200 N-Grams, 0.1 for 400 N-Grams, and 0.101 for 600 N-Grams was given by the SPegasos classifier (Fig. 6c). Receiver Operating Characteristics (ROC) curves is mainly used to compare the classification capability of the different algorithms. Among the number of classifiers tested in this work, it was observed that SPegasos classifier attained the best results.

Similarly, in the *second experiment*, N-Gram of length four bytes was analyzed, and the results for highest accuracy

[1] https://github.com/rieck/malheur/tree/master/data

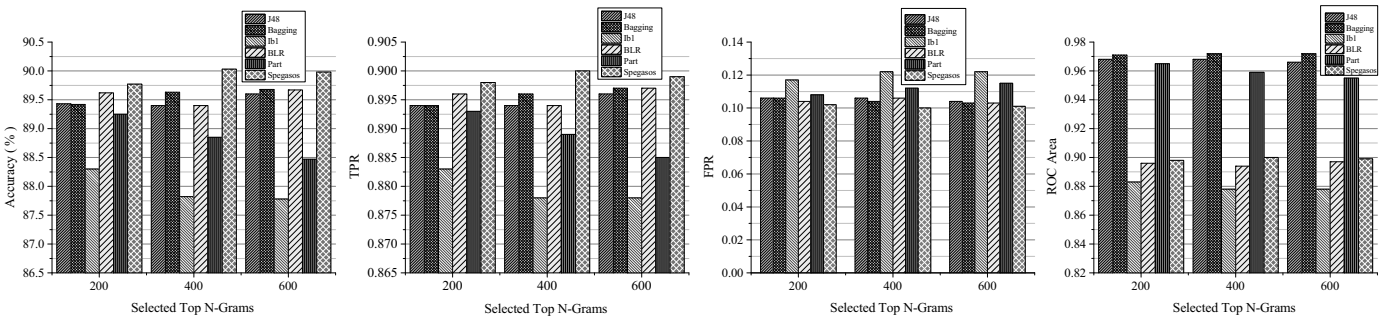
TABLE I: WEKA Classification results for N-Gram Length **3** bytes.

	N-Gram Length= 3 Selected Top N-Grams = 200						**N-Gram Length= 3 Selected Top N-Grams = 400**						**N-Gram Length= 3 Selected Top N-Grams = 600**						
Classifier	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**	
TPR	0.894	0.902	0.881	0.912	0.899	0.896	0.882	0.894	0.874	0.903	0.877	0.886	0.882	0.904	0.874	0.908	0.888	0.874	***B***
	0.895	0.887	0.885	0.88	0.886	0.899	0.906	0.899	0.882	0.885	0.9	0.915	0.91	0.89	0.882	0.885	0.881	0.923	***M***
	0.894	0.894	0.883	0.896	0.893	0.898	0.894	0.896	0.878	0.894	0.889	0.9	0.896	0.897	0.878	0.897	0.885	0.899	**W**
FPR	0.105	0.113	0.115	0.12	0.114	0.101	0.094	0.101	0.118	0.115	0.1	0.085	0.09	0.11	0.118	0.115	0.119	0.077	***B***
	0.106	0.098	0.119	0.088	0.101	0.104	0.118	0.106	0.126	0.097	0.123	0.114	0.118	0.096	0.126	0.092	0.112	0.126	***M***
	0.106	0.106	0.117	0.104	0.108	0.102	0.106	0.104	0.122	0.106	0.112	0.1	0.104	0.103	0.122	0.103	0.115	0.101	**W**
Precision	0.895	0.888	0.885	0.884	0.888	0.899	0.903	0.898	0.881	0.887	0.897	0.912	0.908	0.891	0.881	0.888	0.882	0.919	***B***
	0.894	0.9	0.881	0.909	0.897	0.897	0.885	0.894	0.875	0.901	0.88	0.889	0.885	0.903	0.875	0.906	0.887	0.88	***M***
	0.894	0.894	0.883	0.897	0.893	0.898	0.894	0.896	0.878	0.894	0.889	0.901	0.896	0.897	0.878	0.897	0.885	0.9	**W**
Recall	0.894	0.902	0.881	0.912	0.899	0.896	0.882	0.894	0.874	0.903	0.877	0.886	0.882	0.904	0.874	0.908	0.888	0.874	***B***
	0.895	0.887	0.885	0.88	0.886	0.899	0.906	0.899	0.882	0.885	0.9	0.915	0.91	0.89	0.882	0.885	0.881	0.923	***M***
	0.894	0.894	0.883	0.896	0.893	0.898	0.894	0.896	0.878	0.894	0.889	0.9	0.896	0.897	0.878	0.897	0.885	0.899	**W**
F-measure	0.894	0.895	0.883	0.898	0.893	0.898	0.893	0.896	0.878	0.895	0.887	0.899	0.894	0.898	0.877	0.898	0.885	0.896	***B***
	0.894	0.893	0.883	0.894	0.892	0.898	0.895	0.897	0.879	0.893	0.89	0.902	0.897	0.896	0.878	0.895	0.884	0.901	***M***
	0.894	0.894	0.883	0.896	0.892	0.898	0.894	0.896	0.878	0.894	0.888	0.9	0.896	0.897	0.878	0.897	0.885	0.899	**W**
ROC Area	0.968	0.971	0.883	0.896	0.965	0.898	0.968	0.972	0.878	0.894	0.959	0.9	0.966	0.972	0.878	0.897	0.955	0.899	***B***
	0.968	0.971	0.883	0.896	0.965	0.898	0.968	0.972	0.878	0.894	0.959	0.9	0.966	0.972	0.878	0.897	0.955	0.899	***M***
	0.968	0.971	0.883	0.896	0.965	0.898	0.968	0.972	0.878	0.894	0.959	0.9	0.966	0.972	0.878	0.897	0.955	0.899	**W**
Accuracy (%)	89.43	89.42	88.30	89.62	89.25	***89.77***	89.40	89.63	87.82	89.40	88.85	***90.03***	89.60	89.68	87.78	89.67	88.47	***89.88***	
TPR: True Positive Rate, FPR: False Positive Rate, C1: J48, C2: Bagging, C3: Ib1, C4: Bayesian Logistic Regression, C5: Part, C6: Spegasos, B: Benign, M: Malware, W: Weighted Average																			

TABLE II: WEKA Classification results for N-Gram Length **4** bytes.

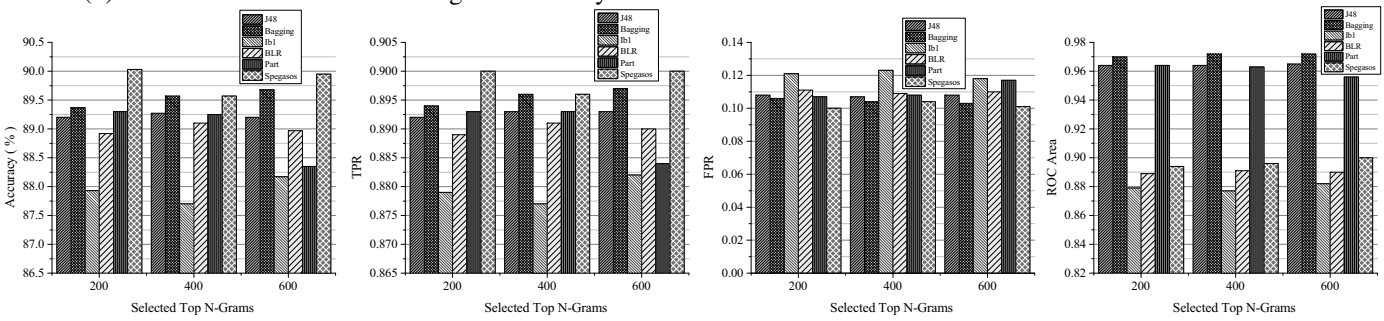
	N-Gram Length = 4 Selected Top N-Grams = 200						**N-Gram Length = 4 Selected Top N-Grams = 400**						**N-Gram Length = 4 Selected Top N-Grams = 600**						
Classifier	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**	
TPR	0.899	0.902	0.88	0.9	0.899	0.921	0.879	0.904	0.881	0.904	0.885	0.9	0.881	0.907	0.881	0.903	0.88	0.894	***B***
	0.885	0.885	0.878	0.878	0.887	0.88	0.907	0.887	0.873	0.878	0.9	0.891	0.904	0.887	0.882	0.877	0.887	0.905	***M***
	0.892	0.894	0.879	0.889	0.893	0.9	0.893	0.896	0.877	0.891	0.893	0.896	0.893	0.897	0.882	0.89	0.884	0.9	**W**
FPR	0.115	0.115	0.122	0.122	0.113	0.12	0.093	0.113	0.127	0.122	0.1	0.109	0.096	0.113	0.118	0.123	0.113	0.095	***B***
	0.101	0.098	0.12	0.1	0.101	0.079	0.121	0.096	0.119	0.096	0.115	0.1	0.119	0.093	0.119	0.097	0.12	0.106	***M***
	0.108	0.106	0.121	0.111	0.107	0.1	0.107	0.104	0.123	0.109	0.108	0.104	0.108	0.103	0.118	0.11	0.117	0.101	**W**
Precision	0.886	0.884	0.879	0.881	0.889	0.887	0.904	0.889	0.874	0.881	0.898	0.892	0.901	0.889	0.882	0.88	0.886	0.904	***B***
	0.898	0.918	0.88	0.898	0.898	0.9	0.882	0.902	0.88	0.902	0.887	0.899	0.884	0.905	0.881	0.9	0.881	0.895	***M***
	0.892	0.901	0.879	0.889	0.893	0.894	0.893	0.896	0.877	0.891	0.893	0.896	0.893	0.897	0.882	0.89	0.884	0.9	**W**
Recall	0.899	0.921	0.88	0.9	0.899	0.902	0.879	0.904	0.881	0.904	0.885	0.9	0.881	0.907	0.881	0.903	0.88	0.894	***B***
	0.885	0.88	0.878	0.878	0.887	0.885	0.907	0.887	0.873	0.878	0.9	0.891	0.904	0.887	0.882	0.877	0.887	0.905	***M***
	0.892	0.9	0.879	0.889	0.893	0.894	0.893	0.896	0.877	0.891	0.893	0.896	0.893	0.897	0.882	0.89	0.884	0.9	**W**
F-measure	0.893	0.902	0.879	0.89	0.894	0.895	0.891	0.897	0.877	0.892	0.892	0.896	0.891	0.898	0.882	0.891	0.883	0.899	***B***
	0.891	0.898	0.879	0.888	0.892	0.893	0.894	0.895	0.877	0.89	0.893	0.895	0.894	0.896	0.882	0.888	0.884	0.9	***M***
	0.892	0.9	0.879	0.889	0.893	0.894	0.893	0.896	0.877	0.891	0.892	0.896	0.892	0.897	0.882	0.89	0.883	0.899	**W**
ROC Area	0.964	0.97	0.879	0.889	0.964	0.894	0.964	0.972	0.877	0.891	0.963	0.896	0.965	0.972	0.882	0.89	0.956	0.9	***B***
	0.964	0.97	0.879	0.889	0.964	0.894	0.964	0.972	0.877	0.891	0.963	0.896	0.965	0.972	0.882	0.89	0.956	0.9	***M***
	0.964	0.97	0.879	0.889	0.964	0.894	0.964	0.972	0.877	0.891	0.963	0.896	0.965	0.972	0.882	0.89	0.956	0.9	**W**
Accuracy (%)	89.20	89.37	87.93	88.92	89.30	***90.03***	89.27	89.57	87.7	89.1	89.25	***89.57***	89.20	89.68	88.17	88.97	88.35	***89.95***	
TPR: True Positive Rate, FPR: False Positive Rate, C1: J48, C2: Bagging, C3: Ib1, C4: Bayesian Logistic Regression, C5: Part, C6: Spegasos, B: Benign, M: Malware, W: Weighted Average																			

were 90.03% for 200 N-Grams, 89.57% for 400 N-Grams, and 89.95% for 600 N-Grams with respect to the SPegasos classifier (Fig. 7a). The highest TPR was 0.9 for 200 N-Grams, 0.896 for 400 N-Grams, and 0.9 for 600 N-Grams obtained by the SPegasos classifier (Fig. 7b). The lowest FPR was 0.1 for 200 N-Grams, 0.104 for 400 N-Grams, and 0.101 for 600 N-Grams produced by the SPegasos classifier (Fig. 7c). From the visual inspection of Fig. 6 and Fig. 7, we can conclude that SPegasos classifier turned out to be best and ensured better classification for both N-Gram lengths three and four.



(a) Accuracy. (b) TPR. (c) FPR. (d) ROC Area.

Fig. 6: Graphical representation considering evaluation measures such as (a) Accuracy, (b) True Positive Rate, (c) False Positive Rate and (d) ROC area. When N-Gram length is three bytes.



(a) Accuracy. (b) TPR. (c) FPR. (d) ROC Area.

Fig. 7: Graphical representation considering evaluation measures such as (a) Accuracy, (b) True Positive Rate, (c) False Positive Rate and (d) ROC area. When N-Gram length is four bytes.

VI. CONCLUSION

In order to detect the malicious activities of the malware, behavior analysis of the executable file (process) such as system calls invoked by the input file during execution have been employed. The gathered system calls' sequence chunked into N-Gram and each N-Gram treated as a feature. The IG feature selection method was used to choose the best features based on highest IG score, and the selected features were used to prepare FFV needed by the classifier. The experiments were performed using different classifiers available in the WEKA tool. From the experimental observations, it was found that the better classifier among the chosen six classifiers in this experimental work is the SPegasos since it achieved highest accuracy, highest TPR, and lowest FPR compared to the others. SPegasos achieved better detection rate for different feature lengths of 200, 400, and 600. Our future work will aim to develop a multiprocessing model able to compute IG scores for larger N-Gram datasets.

REFERENCES

[1] A. Shabtai, R. Moskovitch, Y. Elovici, and C. Glezer, "Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey," *Information Security Technical Report*, vol. 14, no. 1, pp. 16–29, 2009.

[2] Anubis: Analyzing Unknown Binaries-http://analysis.iseclab.org/.

[3] Cuckoo Sandbox-https://cuckoosandbox.org/.

[4] K. Rieck, P. Trinius, C. Willems, and T. Holz, "Automatic analysis of malware behavior using machine learning," *Journal of Computer Security*, vol. 19, no. 4, pp. 639–668, 2011.

[5] C. Willems, T. Holz, and F. Freiling, "Toward automated dynamic malware analysis using cwsandbox," *IEEE Security and Privacy*, vol. 5, no. 2, pp. 32–39, 2007.

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

[7] M. Neugschwandtner, C. Platzer, P. M. Comparetti, and U. Bayer, "Danubis–dynamic device driver analysis based on virtual machine introspection," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2010, pp. 41–60.

[8] Y. Qiao, Y. Yang, J. He, C. Tang, and Z. Liu, "Cbm: free, automatic malware analysis framework using api call sequences," in *Knowledge Engineering and Management*. Springer, 2014, pp. 225–236.

[9] J. Shi, Y. Yang, C. Li, and X. Wang, "Spems: A stealthy and practical execution monitoring system based on vmi," in *International Conference on Cloud Computing and Security*. Springer, 2015, pp. 380–389.

[10] D. K. S. Reddy and A. K. Pujari, "N-gram analysis for computer virus detection," *Journal in Computer Virology*, vol. 2, no. 3, pp. 231–239, 2006.

[11] S. Jain and Y. K. Meena, "Byte level n–gram analysis for malware detection," in *Computer Networks and Intelligent Computing*. Springer, 2011, pp. 51–59.

[12] H. Parvin, B. Minaei, H. Karshenas, and A. Beigi, "A new n-gram feature extraction-selection method for malicious code," in *International Conference on Adaptive and Natural Computing Algorithms*. Springer, 2011, pp. 98–107.

[13] G. J. Tesauro, J. O. Kephart, and G. B. Sorkin, "Neural networks for computer virus recognition," *IEEE expert*, vol. 11, no. 4, pp. 5–6, 1996.

[14] J. Z. Kolter and M. A. Maloof, "Learning to detect and classify malicious executables in the wild," *Journal of Machine Learning Research*, vol. 7, no. Dec, pp. 2721–2744, 2006.

[15] K. Rieck, T. Holz, C. Willems, P. Düssel, and P. Laskov, "Learning and classification of malware behavior," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2008, pp. 108–125.