Detection of Malware using Machine Learning based on Operation Code Frequency

Pavitra Mohandas [1]
Department of Computer Science and Engineering
Sri Venkateswara College of Engineering
(Affiliated to Anna University)
Chennai, India
pavit939@gmail.com

Sudesh Kumar Santhosh Kumar [1]
Department of Electronics and Communication Engineering
Sri Venkateswara College of Engineering
(Affiliated to Anna University)
Chennai, India
sudeshmechanical@gmail.com

Sandeep Pai Kulyadi [1]
Department of Electronics and Communication Engineering
Sri Venkateswara College of Engineering
(Affiliated to Anna University)
Chennai, India
sandeeppai2k@gmail.com

M J Shankar Raman
Department of Computer Science and Engineering
Pravartak Technologies Foundation, Indian Institute of Technology, Madras
Chennai, India
mjsraman@cse.iitm.ac.in

Vasan V S
Department of Computer Science and Engineering
RISE Lab, Indian Institute of Technology, Madras
Chennai, India
vasan.vs@gmail.com

Balaji Venkataswami
Department of Computer Science and Engineering
RISE Lab, Indian Institute of Technology, Madras
Chennai, India
balajivenkat99@yahoo.com

[1] – *contributed equally*

2021 IEEE International Conference on Industry 4.0, Artificial Intelligence, and Communications Technology (IAICT) | 978-1-6654-4494-1/21/$31.00 ©2021 IEEE | DOI: 10.1109/IAICT52856.2021.9532521

***Abstract*— One of the many methods for identifying malware is to disassemble the malware files and obtain the opcodes from them. Since malware have predominantly been found to contain specific opcode sequences in them, the presence of the same sequences in any incoming file or network content can be taken up as a possible malware identification scheme. Malware detection systems help us to understand more about ways on how malware attack a system and how it can be prevented. The proposed method analyses malware executable files with the help of opcode information by converting the incoming executable files to assembly language thereby extracting opcode information (opcode count) from the same. The opcode count is then converted into opcode frequency which is stored in a CSV file format. The CSV file is passed to various machine learning algorithms like Decision Tree Classifier, Random Forest Classifier and Naive Bayes Classifier. Random Forest Classifier produced the highest accuracy and hence the same model was used to predict whether an incoming file contains a potential malware or not.**

Keywords—Malware Analysis, Machine Learning, Metamorphic Malware, Opcode Frequency, Malware Detection

I. Introduction

Malware is a computer program which is specifically built to invade and defile a system without the possessor's authorization. There are various malware including viruses, worms, trojans, spyware, ransomware, etc. Various antivirus companies tend to come across many such malware on an everyday basis, which makes detection of malware an arduous task.

There are numerous malware detection techniques such as behaviour based detection, signature based detection, and machine learning based detection. The major drawback of the signature based detection system is its failure to detect unknown malware. As far as behaviour based detection is concerned, when an antivirus program discerns an attempt to alter a file, a warning will be initiated. Yet there is a chance for a false positive rate. A very comprehensive literature survey has been accomplished and three concise research gaps have been spotted.

a) Signature based detection system fails due to the presence of unknown malware.

b) It is essential to execute malware in behaviour and anomaly based detection systems.

c) Cuckoo Sandbox, a widely used open source malware analysis tool is unable to identify if a malware executable file is embedded in a PDF file.

Hence, this research proposes the design and establishment of opcode frequency based predictor for bridging these gaps by ensuring not to execute the malware and therefore preventing the infection of the environment. The identification of embedded malware files will further improve the standard in identifying masked malware thereby preventing any attack.

II. Objective

The primary objective of this work is to examine the opcodes of portable executable files and find unique opcodes which will be helpful in discovering unknown malware. It is purely based on a machine learning oriented malware detection approach. This approach will aid antivirus firms for identifying unknown and metamorphic malware. It will also minimize the time taken for creating a signature for the malware. The main objective of the work is given below.

a) To inspect and examine different malware detection techniques.

b) To develop an opcode frequency based malware detection approach for detecting metamorphic malware.

c) To authenticate the proposed approach.

III. Related Work

Even though not extensively executed, the idea of using machine learning methods for malware detection is not new. After various types of studies, accuracies of diverse methods were identified.

The technique of malware detection was analyzed in [1] by means of analysis carried out statistically for frequency distribution of opcodes. Statistically 67 malware files were disassembled. The comparison is carried out for opcode frequency distribution between 67 malware files and 20 benign files. With the help of various methods such as Post-hoc Standardized testing, Pearsons Chi Square procedure and Crammers'V statistical analysis for opcode distribution were carried out. It was identified that the frequency distribution of opcodes vary remarkably for benign and malware files.

A detection system was developed in [2] which was formulated on various improved perceptron algorithms. The accuracy ranged between 69.90 percent and 96.18 percent for the diverse algorithms the author had worked on. It is substantial to note that algorithms with highest accuracy had the largest False Positive Rate (48 files). An algorithm which had a low false positive rate had an accuracy of 93.01 percent.

A technique for the detection of behavioural based malware with the help of machine learning was presented in [3] since malware analyses performed manually were ineffective. A behaviour report is produced by a sandbox and prior processing is done by means of Sparse Vector Model. The machine learning classifier was fed with the processed report for classification. The training was performed for various classifiers such as J48 Decision Tree (DT), Naive Bayes, Multilayer Perceptron Neural Network and K-Nearest Neighbours (KNN). J48 Decision Tree outperformed the other algorithms and provided an accuracy of 96.80 percent.

The work done in [4] deduced that supervised machine learning approach can be used to discern unknown potential malware but it is essential to have copious malignant and benign executable files and labelled data. They have devised a semi - supervised machine learning based approach for the detection of unknown malicious files. The technique was on the basis of examining the presence of frequency of opcode sequences. They had manifested that the above approach required fewer labelled data as compared to supervised machine learning approach.

It was determined in [5] that from Microsoft portable executable file format, seven fundamental attributes in order to classify any incoming file as either malware or benign with the help of a machine learning model was identified. Hence, the void is infused in the middle of the incoming malware in the system and detection of malware using signature based methods. In order to classify any incoming file to be a malware file or benign file with the help of a machine learning model, the seven identified attributes from Microsoft portable executable file format are ImageVersion, ExportSize, VirtualSize2, DebugSize, IatRVA, ResourceSize and NumberOfSections.

An approach was demonstrated in [6] which detects metamorphic malware by using resemblance of executable files formulated on opcode graphs. From the assembly of a program, opcodes are extracted initially and a weighted opcode graph is designed. Every node consists of a distinct opcode and a successor opcode is connected by an edge. The frequency of occurrence of the successor opcode is given as a weight to an edge. The graph that is generated is collated with a known malware's graph using matrices. The collation is obtained from an evaluation function which was established in their paper. A malware is identified if the similarity score of the correlation is less than the threshold value else it is considered to be benign.

A detection technique was put forward in [7] which focused on segregating malware and benign programs with the help of a classifier which is a solo model. The classifier that has been trained strives to differentiate the behavior of all benign programs from all other malicious programs. For a specific program, its implementation runs could be either benign or malicious based on whether the malware injected is triggered, making it arduous to label the program while training. The classifier is susceptible to the choice of benign and malicious examples in the training set and might ensue in unacceptable false positives and false negatives.

A method for detection of malware in android system was proposed in [8]. The features for detection were considered to be Permission and API Calls. Feature selection was carried out by Relief algorithm. Three machine learning classifiers namely J48, SVM and Naive Bayes were trained. The obtained rate of detection is good. But the computation load and the amount of resources used is very high. Our work is more comprehensive and can be used in any kind of system while [8] is restricted to android system.

A technique for classifying executable files using machine learning and frequency of opcodes was proposed in [9]. For the purpose of feature selection, IDA pro disassembler has been used to disassemble the executable files. In our work, we have used "Objdump" disassembler which is a command-line tool that can be easily used to disassemble executable files in real time during the malware detection process. In their work, the dataset size was small where a total of only 100 malware and benign files were used where the malware samples were downloaded from a single source. In our experiment, a dataset of about 250 files which included malware samples from three different sources were used. In addition to this, they have used only 20 frequent opcodes as features and have fed them to the machine learning classifiers like SVM, BOOSTING, Random Forest and Decision Tree for the purpose of classification of executable files as malware or benign.

An amalgamated approach for detection of malware in Android by using distinct sequential system call and system call was proposed in [10]. The behaviours of unknown applications were identified at a detection rate of 91.76 percent. However, real time detection is not supported which is achieved in our work with a higher detection rate. The malware detection method proposed in [11] uses machine learning techniques centered on the amalgamation of dynamic and static features. In their work, they have extracted the static features like strings and DLL's in executable files using the IDA python plugin. In addition to this they have also extracted dynamic features like assembly instruction sequences and API call frequency. After the feature selection process, they fed the input features to Naive Bayes classifier and SVM classifier.

The paper [12] is based on a survey on visualization, detection and classification of malicious software. This work uses techniques that do not perform the disassembling process to get the assembly instruction of files. An executable file represented in the form of a string of binaries (zeros and ones). It also presents an extensive study of classification and visualization of malware images. Computer vision techniques are employed and malware is depicted as images in which different binary sections are studied. However, in our case, the process of disassembling and the usage of assembly instructions promise better results.

IV. Basic Definitions and Concepts

A. Metamorphic Malware

A metamorphic malware is one which can transmute depending on the potential to interpret, emend and rephrase its original code. It is contemplated as the most treacherous computer virus and it has the capability to do significant detriment to a system if it is not discerned quickly. Anti-virus scanners have an onerous time recognizing this type of virus because it can vary its internal formation by rephrasing and reprogramming itself each and every time it taints a computing system. A metamorphic malware is divergent from a polymorphic virus, which encrypts its primal code to keep away from being discerned. Because of the intricacies of metamorphic viruses, it involves immense programming skills.

TABLE I. DIFFERENCES BETWEEN POLYMORPHIC AND METAMORPHIC MALWARE

Differences between Polymorphic Malware and Metamorphic Malware	
Polymorphic Malware	***Metamorphic Malware***
Polymorphic malware are effortless to program.	Metamorphic malware are strenuous to program.
Polymorphic malware encrypt them with a variable encryption key so that each copy of the virus appears different.	Metamorphic malware rephrase their primal codes to make them appear unalike each and every time.

B. Operation Code (Opcode)

In computer engineering, an Operation Code abbreviated as Opcode is the chunk of a machine language instruction which describes the function to be carried out. An opcode is the initial instruction and it is also called as instruction code or opstring.

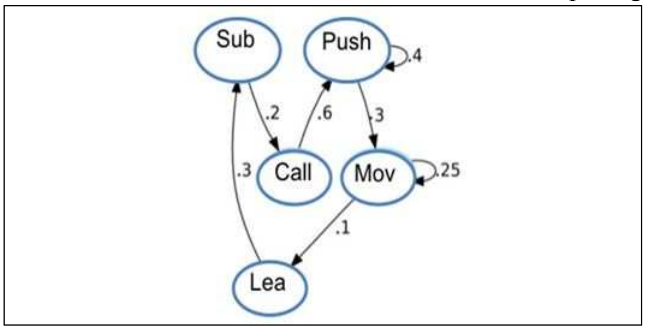

Fig.1. Functioning of opcodes

V. Proposed Method

The major hindrance of transformation is still preserved as defiance with ample ventures of malware developers. The frequency of opcode occurrences is taken into consideration as an indicator for differentiating malware files from benign files. The major objective of our method is in line with the detail that there are typical features which are common amidst the produced mutates of a metamorphic engine. Fig. 2 shows the various steps in the proposed method.

A. Steps of proposed method

Step 1: From the client system, the user requests for a file from the Flask web server in the bash shell. While requesting, filename along with its extension is passed in the command-line to the client system.

Step 2: The received filename is sent in json format to the endpoint of the server which checks if the file is present in the server or not. This endpoint returns “yes” if the file is present in the server or it returns “no” if the file is not present to the client system.

Step 3: If the file is present, the filename is sent to another endpoint of the server. In this endpoint, there are four steps. They are,

Step i]: The requested PDF file is checked for any embedded executable files within it. The embedded executable file (.exe) is extracted as “embed.exe” and saved in a directory.

Step ii]: The codes of the executable file under survey (including both malicious and benign files) in the directory of executable files is converted to assembly code (.asm) using a disassembler. A disassembler like “Objdump” is used in this step. Objdump is a command-line program for disassembling the executable files in a UNIX-like operating system. The generated assembly code (.asm) is stored in a directory.

Step iii]: The assembly code that is generated in the above step is converted to opcode frequencies and stored in the form of a spreadsheet (.csv) file.

Step iv]: A trained Random Forest model is used to make predictions by passing the opcode frequencies as features. Based on the predictions of the machine learning algorithm, the executable file is predicted as a benign or malware file.

Step 4: If the PDF document requested by the user has a potential malware embedded within it, the user is prompted with a warning message and the process will be interrupted. In case the file requested by the user is benign, the pdf document will be sent to the user in zip format and will be saved in the directory where the user wanted to save the file.

B. System Architecture

Based on signature based methods and heuristic approach, prevalent malware detection systems were designed. It was liable to commence novelty in this field since existing systems were unable to perceive metamorphic malware.As per ongoing research, with the help of statistical metrics, schemes of metamorphic malware can be identified. In order to generate mutations of a malware, a single engine is used even though a malware has different mutations. Eventually, there is a prospect for observing analogous statistical schemes.

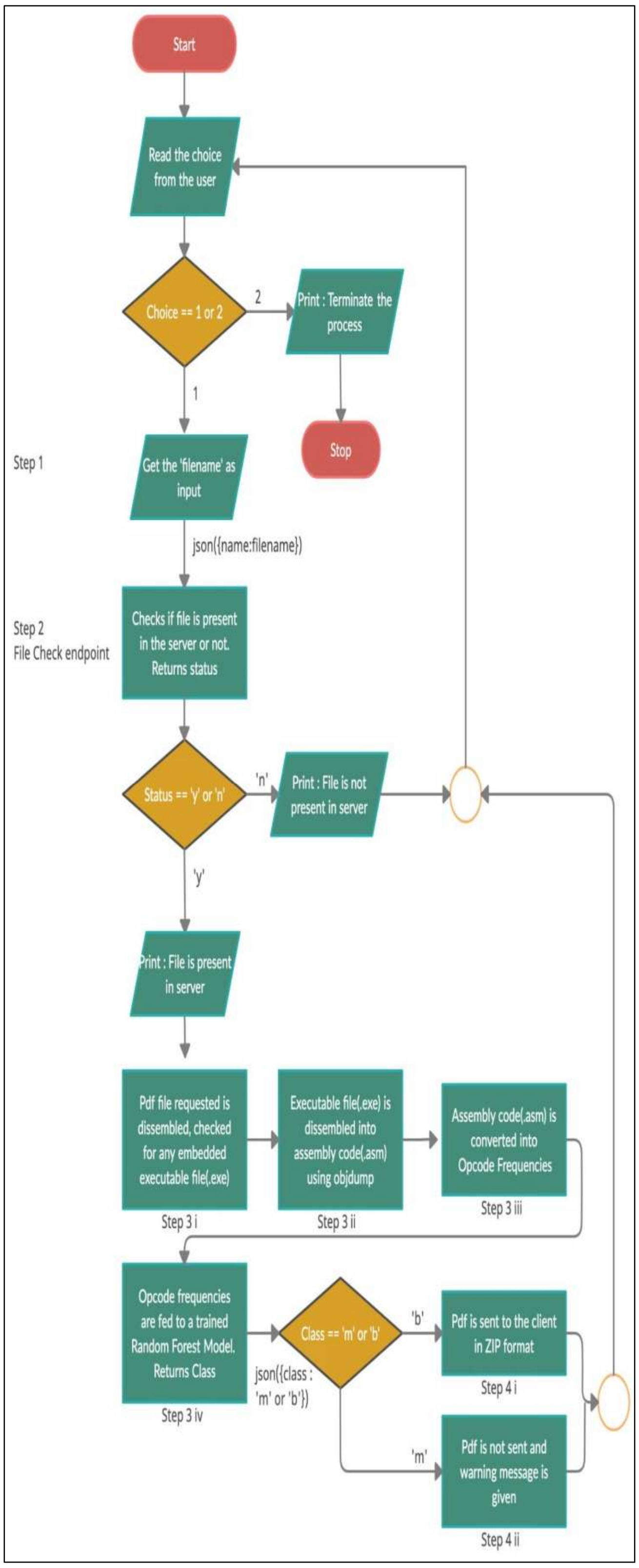
Start
Read the choice from the user
Choice == 1 or 2
2
Print : Terminate the process
Stop
1
Step 1
Get the 'filename' as input
json({name:filename})
Step 2
File Check endpoint
Checks if file is present in the server or not. Returns status
Status == 'y' or 'n'
'n'
Print : File is not present in server
'y'
Print : File is present in server
Pdf file requested is dissembled, checked for any embedded executable file(.exe)
Step 3 i
Executable file(.exe) is dissembled into assembly code(.asm) using objdump
Step 3 ii
Assembly code(.asm) is converted into Opcode Frequencies
Step 3 iii
Opcode frequencies are fed to a trained Random Forest Model. Returns Class
Step 3 iv
Class == 'm' or 'b'
json({class : 'm' or 'b'})
'b'
Pdf is sent to the client in ZIP format
Step 4 i
'm'
Pdf is not sent and warning message is given
Step 4 ii

Fig. 2. Steps of the Malware Detection Process

C. Training Dataset

The training dataset includes two sets of executable files. The benign class includes 100 benign (healthy) executable files that are collected by using Windows applications in the Program Files directory. The malware class consists of 150 metamorphic malware executable files collected from various websites. Table II represents the training data collected from various sources.

TABLE II. SOURCES OF TRAINING DATASET

S NO	**TRAINING DATASET**		
	Description	***Number of files***	***Source***
1.	Benign Executable Files	100	Windows Applications
2.	Malware Executable Files	50	Malware Bazaar Website
3.	Malware Executable Files	50	Tek Defence Website
4.	Malware Executable Files	50	Malshare Website

D. Feature Extraction

In artificial intelligence, a feature is a quantifiable entity of an event being examined. Generally in datasets, features are present as columns. Features are discrete self-sufficient variables that act like inputs to a system. Actually, during the process of training, models make use of features to compute the predictions. With the help of feature engineering processes, new features could be derived from old features. In machine learning and Knowledge Discovery in Data (KDD), one of the conspicuous issues is feature selection.

The opcodes extracted from the assembly codes (.asm) generated by disassembling executable files (.exe) are the features. Table III shows the various types of opcodes with examples. In this research, around 953 opcodes were scrutinized. Every single assembly command comprises two parameters. They are the opcode and the operands or registers. For the above process, only opcodes are substantial and the operands are no longer necessary. Hence, the operands in every assembly instruction are eliminated and the opcodes are retained.Once the opcodes are extracted, the next step would be to calculate the number of times every opcode appears in a particular assembly code (.asm). The opcodes and their respective number of occurrences for every file in the assembly codes directory are generated as a spreadsheet (.csv).

Each and every metamorphic malware executable file or benign executable file (Windows Applications) might be fabricated using a diverse number of opcodes. Hence, the count of opcodes in every training example might be dissimilar with each other. In order to subsist with undue numbers of opcodes in every sample, the necessity to reckon the opcode frequencies arises. The frequency of a particular opcode is computed by dividing the number of occurrences of a specific opcode by the entire number of opcodes in the assembly code (.asm). (1) demonstrates the calculation of opcode frequencies.

$$\text{Opcode Frequency} = \frac{\text{No. of Instances of the Opcode}}{\text{Total No of Opcodes in the files}} \qquad (1)$$

TABLE III. PATTERNS OF OPCODES USED IN THE PROPOSED METHOD

Types and examples of opcodes utilised in the proposed method	
Group	***Sample***
Arithmetic Instructions	ADC, SBB, DEC
Logical Instructions	XRA, CMP, ANA
Data Transfer Instructions	MOV, LAHF, OUT
Jump Instructions	JMP, JE, JC
Instructions Involving The Stack	PUSH, POP, CALL
Instructions Converting Dimensions of Data	CDQ, CBW, CWD
Instructions Performing Comparison Operations	CMP, CPI, SCAS
Instructions Involving Rotation of Data	RLC, RAR, RRC
Other Instructions	WAIT, NOP, REP

VI. SYSTEM REQUIREMENT

The system which is proposed was executed in "Ubuntu 18.04" operating system running on top of an "Intel core i5 10th generation" processor with 512 GB SSD and 8 GB RAM. The programming language employed for the development of the proposed system was "Python version 3.6.9" along with "Flask version 1.1.2".

VII. IMPLEMENTATION

The malware executable files downloaded from various websites were fed to Cuckoo Sandbox, an open source malware detection platform. It was able to predict the provided file to be a potential malware.

When the same executable files were embedded into PDF files and fed back to Cuckoo Sandbox, it predicted the files to be benign (healthy) in spite of malware being present inside the PDF files. Having identified this vulnerability, a system with the ability to discern masked malware was built.

The initial step was to create a data repository consisting of PDF files embedded with benign and malware executable files. Once this is done, a web server which can process requests from various clients was established using the Flask framework in order to demonstrate the process of metamorphic malware detection.

The steps incorporated in the proposed method such as disassembling of a PDF file to check for embedded executable files, disassembling of executable files (.exe) to assembly code (.asm), creating an opcode frequency spreadsheet (.csv) from the generated assembly code and prediction of the embedded executable file to be malware or benign was carried out.

In order to perform the prediction process, various machine learning algorithms such as the Random Forest Classifier [13], Naive Bayes classifier and Decision Tree classifier [14] were trained over the data collected from various sources.

Finally, the client system is deployed using python in order to validate the Flask web server by passing various HTTP requests (GET, POST) to the server. In case the file requested by the client system is predicted to contain a potential malware by the server, a warning message is prompted to the user and the file is prevented from being sent to the client system in order to protect the client system against any malware attack. If the file is predicted to be benign, it is sent successfully to the client system in zip format.

VIII. SYSTEM EVALUATION

Whenever a machine learning based system has to be evaluated, a testing dataset is required which would determine the ability of the system to detect a potential malware when deployed.

The malware detection system was tested for the dataset which contains 100 files including 20 benign (healthy) files, 50 malware files from Malshare website, 20 malware files from Malware Bazaar website and 10 malware files from Tek defence website. Table IV represents the number of files in each class.

TABLE IV. SOURCES OF TESTING DATASET

S.NO	**TESTING DATASET**		
	Description	***Number of files***	***Source***
1.	Benign Executable Files	20	Windows Applications
2.	Malware Executable Files	20	Malware Bazaar Website
3.	Malware Executable Files	10	Tek Defence Website
4.	Malware Executable Files	50	Malshare Website

To evaluate the proposed method, three classification algorithms are used. They are Decision Tree classifier, Naive Bayes classifier and Random Forest classifier. In order to evaluate the performances of the above algorithms, following metrics are used:

True Positives (TP): True Positives is the number of malware executable files that are correctly classified as malware.

True Negatives (TN): True Negatives is the number of benign executable files that are correctly classified as benign.

False Positives (FP): False Positives is the number of benign executable files that are incorrectly classified as malware.

False Negatives (FN): False Negatives is the number of malware executable files that are incorrectly classified as benign.

From the values of True Positives (TP), True Negatives (TN), False Positives (FP) and False Negatives (FN), True Positive Rate (TPR) and False Positive Rate (FPR) are obtained. (2) and (3) show how True Positive Rate and False Positive Rate are calculated. The Total Accuracy is calculated by dividing the number of true predictions by the total detections. (4) shows how Total Accuracy is being calculated.

$$\text{True Positive Rate} = TP/(TP + FN) \quad (2)$$

$$\text{False Positive Rate} = FP/(FP + TN) \quad (3)$$

$$\text{Accuracy} = (TP + TN)/(TP + TN + FP + FN) \quad (4)$$

In addition to the above metrics, the evaluation metrics like Precision, Recall and F1-score are used to evaluate the three models. Precision shows what percentage of the positive predictions is truly correct. Precision is the ratio of True Positives (TP) and all the positives (TP + FP). Recall is the measure of how well the True Positives are correctly identified by the model. F1-score gives the equilibrium between Precision and Recall. It can also be called an F-score or F-measure. (5), (6) and (7) show how the Precision, Recall and F1-score values are calculated, respectively.

$$\text{Precision} = TP / (TP + FP) \quad (5)$$

$$\text{Recall} = TP / (TP + FN) \quad (6)$$

$$\text{F1-score} = 2*((\text{Precision}*\text{Recall})/(\text{Precision} + \text{Recall})) \quad (7)$$

The proposed method is assessed with Accuracy, True Positive Rate (TPR), False Positive Rate (FPR), Precision, Recall and F-1 score.

IX. Accuracy

Fig. 3 shows the accuracies of the three models in the evaluation (testing) phase. With reference to Fig. 3 it is evident that Random Forest classifier has the highest accuracy of 97.97 percent and Decision Tree classifier ranks second with an accuracy of 95.90 percent and Naive Bayes classifier has the least accuracy of 89.80 percent.

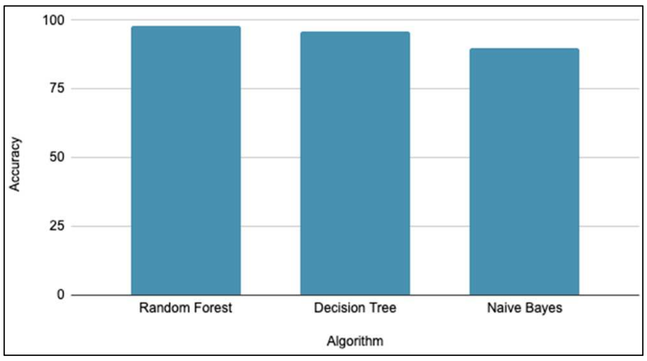

Fig.3. Comparison of accuracies between various machine learning algorithms

Table V shows the performances of the three classifiers based on all the evaluation parameters. From Table V it is apparent that for the Random Forest classifier, TPR is 0.97 and FPR is 0.0. In case of the Decision Tree classifier, TPR is 0.90 and FPR is 0.0 and in case of the Naive Bayes classifier, TPR is 0.96 and FPR is 0.2.

Although the FPR of Decision Tree classifier is 0.0, its TRP is just 0.9 and it is clear that the TPR of Naive Bayes classifier is almost equal to that of the Random Forest classifier but the FPR value is poor. Considering all the evaluation metrics it is very evident that the Random Forest classifier outperforms all the other classifiers with an accuracy of 97.9, TPR is 0.97 and FPR is 0.0.

TABLE V. EVALUATED METRICS SUMMARY

Algorithms	**Summary of All Evaluation Metrics**					
	Accuracy	***TPR***	***FPR***	***Precision***	***Recall***	***F1-Score***
Random Forest	97.97 %	0.97	0.0	0.9767	0.9827	0.9793
Decision Tree	94.9%	0.90	0.0	0.6587	0.6551	0.6568
Naive Bayes	89.89%	0.96	0.2	0.9089	0.8851	0.8932

X. Conclusion

In this work, we established a new method for detecting unknown and metamorphic malware with the help of opcode frequencies. The detection of malware when embedded into other file formats was also accomplished successfully. Findings indicated that all metamorphic malware were successfully spotted absolutely without any false positives. For system training, three machine learning algorithms such as Decision Tree classifier, Random Forest classifier and Naive Bayes classifier were built. Since the Random Forest model outperformed the other two algorithms it was selected for the system to classify incoming files.

Malware Detection based on opcode frequency is in accordance with assembly instruction analysis. The proposed method for detecting malware is considered to have limited obscurity. Hence, this system can be easily set up for detecting malware and can be infused with existing open source malware detection systems for discerning malware embedded in various file formats.

Acknowledgement

The authors would like to thank Shankar Raman M J and Vasan V S for their guidance and constant support throughout this work. Pavitra Mohandas, Sudesh Kumar Santhosh Kumar and Sandeep Pai Kulyadi contributed equally to this paper.

References

[1] D. Bilar, “Opcodes as predictor for malware,” Int. J. Electron. Secur. Digit. Forensics, vol. 1, no. 2, p. 156, 2007.

[2] D. Gavrilut, M. Cimpoesu, D. Anton, and L. Ciortuz, “Malware detection using machine learning,” in 2009 International

Multiconference on Computer Science and Information Technology, 2009, pp. 735–741.

[3] I. Firdausi, C. Lim, A. Erwin, and A. S. Nugroho, "Analysis of machine learning techniques used in behavior-based malware detection," in 2010 Second International Conference on Advances in Computing, Control, and Telecommunication Technologies, 2010.

[4] I. Santos, J. Nieves, and P. G. Bringas, "Semi-supervised learning for unknown malware detection," in Advances in Intelligent and Soft Computing, Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 415–422.

[5] K. Raman, "Selecting features to classify malware," Infosecsouthwest.com.

[6] N. Runwal, R. M. Low, and M. Stamp, "Opcode graph similarity and metamorphic detection," J. Comput. Virol., vol. 8, no. 1–2, pp. 37–52, 2012.

[7] M. Ozsoy, C. Donovick, I. Gorelik, N. Abu-Ghazaleh, and D. Ponomarev, "Malware-aware processors: A framework for efficient online malware detection," in 2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA), 2015, pp. 651–661.

[8] S. Sheen, R. Anitha, and V. Natarajan, "Android based malware detection using a multifeature collaborative decision fusion approach," Neurocomputing, vol. 151, pp. 905–912, 2015.

[9] A. Yewale and M. Singh, "Malware detection based on opcode frequency," in 2016 International Conference on Advanced Communication Control and Computing Technologies (ICACCCT), 2016, pp. 646–649.

[10] F. Tong and Z. Yan, "A hybrid approach of mobile malware detection in Android," J. Parallel Distrib. Comput., vol. 103, pp. 22–31, 2017.

[11] J. Zhao, S. Zhang, B. Liu, and B. Cui, "Malware detection using machine learning based on the combination of dynamic and static features," in 2018 27th International Conference on Computer Communication and Networks (ICCCN), 2018, pp. 1–6.

[12] R. Komatwar and M. Kokare, "A survey on malware detection and classification," J. Appl. Secur. Res., pp. 1–31, 2020.

[13] L. Breiman, Mach. Learn., vol. 45, no. 1, pp. 5–32, 2001.

[14] L. Rokach and O. Maimon, "Decision Trees," in Data Mining and Knowledge Discovery Handbook, New York: Springer-Verlag, 2006, pp. 165–192.

[15] U. Bayer, A. Moser, C. Kruegel, and E. Kirda, "Dynamic analysis of malicious code," J. Comput. Virol., vol. 2, no. 1, pp. 67–77, 2006.

[16] R. K. Shahzad, N. Lavesson, and H. Johnson, "Accurate adware detection using opcode sequence extraction," in 2011 Sixth International Conference on Availability, Reliability and Security, 2011, pp. 189–195.

[17] N. Usman et al., "Intelligent dynamic malware detection using machine learning in IP reputation for forensics data analytics," Future Gener. Comput. Syst., vol. 118, pp. 124–141, 2021.

[18] F. A. Narudin, A. Feizollah, N. B. Anuar, and A. Gani, "Evaluation of machine learning classifiers for mobile malware detection," Soft Comput., vol. 20, no. 1, pp. 343–357, 2016.

[19] M. Kedziora, P. Gawin, M. Szczepanik, and I. Jozwiak, "Malware detection using machine learning algorithms and reverse engineering of android java code," SSRN Electron. J., 2019.

[20] B. A. Mantoo and S. S. Khurana, "Static, dynamic and intrinsic features based android malware detection using machine learning," in Lecture Notes in Electrical Engineering, Cham: Springer International Publishing, 2020, pp. 31–45.