MALWARE DETECTION IN INTERNET OF THINGS USING OPCODES AND MACHINE LEARNING

by

Aditi Atul Khare
A Thesis
Submitted to the
Graduate Faculty
of
George Mason University
in Partial Fulfillment of
The Requirements for the Degree
of
Master of Science
Computer Engineering

Committee:

_____	Dr. Avesta Sasan, Thesis Director
_____	Dr. Khaled Khasawneh, Committee Member
_____	Dr. Sai Manoj Pudukotai Dinakarrao, Committee Member
_____	Dr. Monson Hayes, Department Chair
_____	Dr. Kenneth S. Ball, Dean, Volgenau School of Engineering
Date:_____	Fall Semester 2020 George Mason University Fairfax, VA

Malware Detection in Internet of Things Using Opcodes and Machine Learning

A Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science at George Mason University

by

Aditi Atul Khare
Bachelor of Engineering
Sir M. Visvesvaraya Institute of Technology, 2017

Director: Dr. Avesta Sasan, Associate Professor
Department of Electrical and Computer Engineering

Fall Semester 2020
George Mason University
Fairfax, VA

DEDICATION

This thesis is dedicated to my loving mother, Mrs. Swati Khare who constantly supported and encouraged me throughout my graduate studies.

ACKNOWLEDGEMENTS

I have come across a lot of people who have played an important role during my graduate studies. I would like to take this opportunity to thank all of them for always having faith in me. I would especially thank Dr. Avesta Sasan for all the support and encouragement that has resulted in me completing my research. I also want to take this moment to thank Dr. Liling Huang and Ms. Jammie Chang for believing in me and helping me through every step of my education.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

Adaptive Boosting ..... AdaBoost
Area Under the Curve ..... AUC
Business-to-Business B2B
Cross Validation CV
Distributed Denial of Service ..... DDoS
False Negative ..... FN
False Positive ..... FP
Frequent Sequence Mining ..... FSM
Internet of Battlefield Things ..... IoBT
Internet of Things ..... IoT
K-Nearest Neighbors ..... KNN
Matthews Correlation Coefficient ..... MCC
Maximal Sequential Pattern ..... MSP
Mind the Gap: Frequent Sequence Mining ..... MG-FSM
Personal Computer ..... PC
Receiver Operating Characteristics ..... ROC
Secure Shell ..... SSH
Support Vector Machine ..... SVM
True Negative ..... TN
True Positive ..... TP

ABSTRACT

MALWARE DETECTION IN INTERNET OF THINGS USING OPCODES AND MACHINE LEARNING

Aditi Atul Khare, M.S.

George Mason University, 2020

Thesis Director: Dr. Avesta Sasan

In the recent years, the exponential growth of Internet of Things devices has caused a huge security threat. These devices are being deployed even before being secured. Most of the IoT devices are either unsecured or weakly secured and attackers are taking advantage of this. Even if one IoT device gets infected, it has the potential to spread the malware to the entire network. Obfuscation techniques like polymorphism are being used by hackers to avoid detection.

This research is focused on polymorphic malware detection in Internet of Things networks using opcodes and machine learning. ARM-based malware was used for testing because of the large share of ARM-based IoT platforms making it more indictive of real-world attacks. Opcodes were extracted by disassembling the dataset using the IDA Pro disassembler. A sequentially ordered dataset of the opcodes was created to be used for detection. Four different datasets namely $D_{malware}$, $D_{goodware}$, $D_{unseenmalware}$ and $D_{unseengoodware}$

were created. A polymorphed version of the unseen malware dataset was also created to test the utility of the approach in polymorphic malware detection. We used the sequential pattern mining algorithm, Mind the Gap: Frequent Sequence Mining, to mine the most frequent patterns in malware. These Maximal Sequential Patterns aka MSPs were categorized based on their functionality using ARM resources.

Three different approaches were tested and compared. The first approach was to create an opcode-rank dictionary based on opcode frequency in the malware dataset to create vectors for machine learning classification. The second approach used the frequency of MSPs to vectorize the given dataset while the third approach used the MSP type as a feature for detection. Machine learning classifiers like Decision tree, KNN, Random-Forest, SVM and AdaBoost were used to detect malware as well as polymorphic malware. It was observed that the sequential pattern mining approaches were faster and more resilient to polymorphed malware. A comparative study showed that the MSP list approach has comparable performance to the MSP type approach. Also, the MSP list approach has faster pre-processing runtimes and lower memory usage making it a viable approach for classification of malware.

INTRODUCTION

Malware in Internet of Things

Malware is essentially malicious software intended to cause harm or access sensitive information from a system or network [1]. Malware is a set of instructions that runs on your system and performs operations as defined by the hacker. A common reason as to why an attacker uses malware can be to extort money by encrypting classified information and asking the victim to pay for the decryption key. Other reasons can include theft of information like bank details or espionage. Stealing personal information is always a priority of malicious programs. Popular platforms with a large user base are often a target for hackers. Research and development of efficient ways to detect malware is imperative as malware is an active threat to individuals, businesses, and the government.

Internet of things is a system of unique smart objects that can sense, compute, and communicate with the ability to transfer data over a network with little to no human to computer interaction. The number of IoT devices is growing quickly and is projected to grow to 22 billion by 2025. Figure 1 shows total number of active device connections worldwide. The non-IoT devices are all mobile phones, tablets, PCs, laptops, and fixed line phones while the IoT devices includes all consumer and B2B connected devices [2]. Rapid developments in IoT networks has made it difficult to secure them prior to their

deployment making them prone to attacks. Since IoT devices can make decisions autonomously, securing them is extremely important. If one device in the network gets infected, it has the potential to propagate the malware across the entire network [3]. Malware threats are always a cause of concern. A new Linux malware called Kaiji was observed by the security firm at Intezer in 2020, to be targeting IoT devices using SSH brute force techniques [4]. Before we detect malware, it is important to understand how malware can be obfuscated by the attacker to avoid detection which will be covered in the following section.

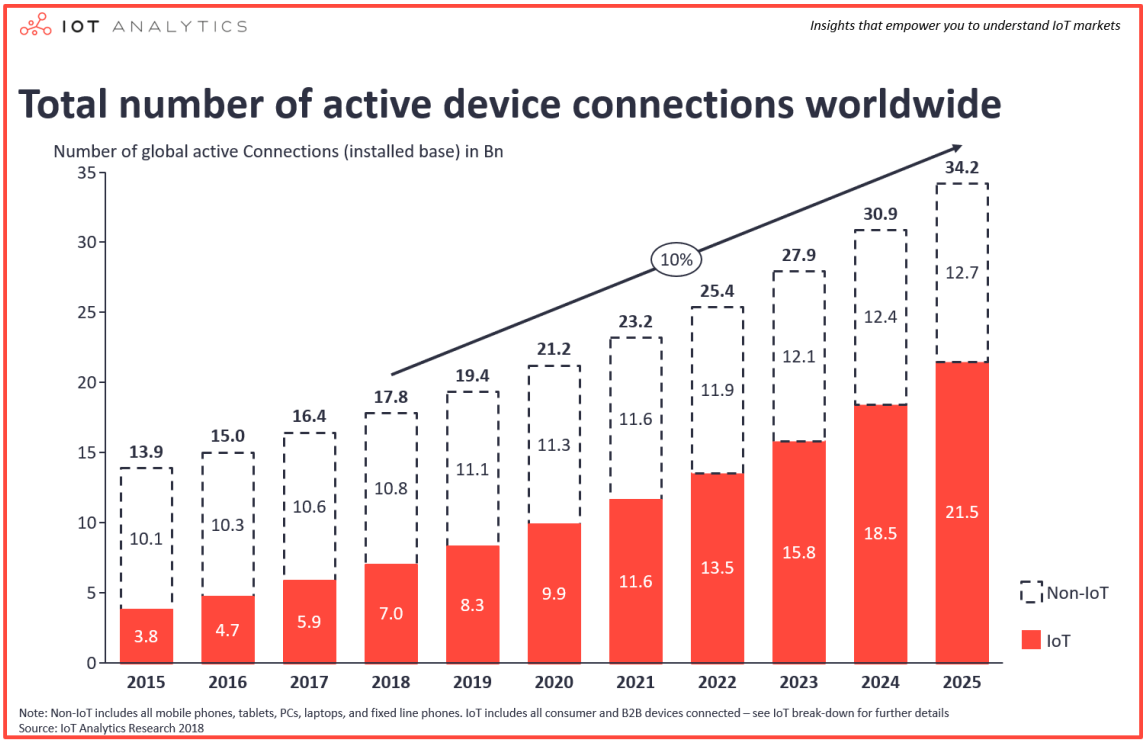

Figure 1 Total Number of Connected IoT Devices

Malware Obfuscation

Malware is used by criminals to steal private and classified information. As malware developers create more sophisticated versions, it is progressively getting more difficult to identify them. Attackers often use different obfuscation techniques to make detection difficult. Obfuscation is also used by companies to protect their source code. An obfuscated program essentially does the same thing as the original code, but it is more difficult to understand.

Various obfuscation techniques are being used like packers, encryption, polymorphic code, and metamorphic code, and these are becoming huge challenges in the field of security. We discuss some of them below.

Methods to obfuscate malware:

1. Encryption: The malicious part of malware is often encrypted. A new random key is generated for every infection. This encrypted part of the code is decrypted only when the file is run. The part of the code that performs the decryption will be the same for all versions thereby enabling the malware to be recognized based on the pattern of the decryption code. [5]

2. Packers: Packers can be used to change the signature of a file. Packers were originally created to decrease the size of the file to cater to the limitations on disk space and bandwidth but are now frequently used by attackers. Packers are great obfuscation tools since a small change will produce a very different signature. [6]

3. Polymorphic: Polymorphism is a technique in which the static binary code is changed. Polymorphic malware constantly changes its identifiable features to avoid pattern-

matching detection. Some characteristics change but the functional purpose remains the same. Common forms of polymorphic malware include viruses, worms, keyloggers, bots and trojans. [7] In our research, we focus on detecting polymorphic malware.

4. Metamorphic: Metamorphic malware is re-written every time so that the new version is different from the previous one. The code changes making it difficult to be detected as the same malicious program by signature-based antivirus software. Even though the code changes, each iteration performs the same function.

Malware Analysis

There are two main approaches that we will discuss here: static or dynamic.

Static analysis relies on features extracted from API calls, strings, byte n-gram**s** and opcodes for malware detection. In this method, the malware is analyzed without executing the code. Malware features can be viewed by reverse engineering the binary file. In our research, we make use of static methods by disassembling the malware samples and analyzing the opcodes.

Dynamic analysis techniques rely on the execution of the given program to detect the malicious run-time features. The malware is run in an isolated, virtual environment for analysis. Using this method, the function and behavior such as transmitted network traffic and length of execution of the malware sample is analyzed for detection. [8]

Importance of Opcodes and Machine Learning in Malware Detection

It has been found that opcodes can be a good indicator for prediction of malware because some opcodes have a higher frequency in malware when compared to benign

samples. [1] Understanding the relevance of the frequent patterns of opcodes in malware is the main motivation of this research.

The quick development in the type and number of malware species has made it extremely difficult to give an on-time reaction. Thus, machine learning aided malware analysis has become a need for automating different parts of static and dynamic analysis. There has been an effort to merge machine learning techniques with malware analysis to speed up and automate malware analysis tasks. [9]

BACKGROUND

Data Mining

Data mining is the process of finding irregularities, patterns, and interconnections within large datasets for prediction of results. It is an interdisciplinary field comprised of statistics, artificial intelligence, and machine learning. Data mining is important since just data has no information until it is structured. Extraction of information is imperative to understand useful data, to make good use of the information and for making informed decisions, among others. [10] Data mining is especially important in machine learning. Machine learning is the design, study and development of algorithms that enable machines to learn and make decisions autonomously without humans. It is a tool to make machines smarter. [11]

Some common data mining techniques are:

1. Classification: involves analyzing the different attributes of data to categorize related data.
2. Association: Analysis of data to extract relationship between data events. Ex. Purchase of burgers is frequently accompanied with that of fries.
3. Outlier detection: this technique mines anomalies in datasets. Useful in understanding why aberrations happen and how to deal with them.

4. Clustering: Relies heavily on visual approaches and is used to show where the distribution of data is with respect to different metrics.
5. Regression: this method reveals how variables are related and are useful in forecasting and data modelling.
6. Prediction: predictive analysis finds patterns in current or historical data to extend them into the future.
7. Sequential patterns: focuses on understanding series of events that takes place in sequence.
8. Decision trees: this technique enables users to understand how the data inputs affect the outputs.
9. Visualization: it gives insights into data to the users by visualizing the different trends and patterns in data by using colors. Dashboards are a great example of this technique. [12]

In our work, we make use of sequential data mining to mine the most frequent patterns of opcodes in the malware dataset.

MG-FSM

Some opcodes have a higher frequency than others in malware. In this research, we mine the maximal sequential patterns (MSPs) and their frequencies to determine if a given sample is malware or not. To find these MSPs, we make use of Mind the Gap: Frequent Sequence Mining algorithm.

FSM can be used for statistical models, information retrieval, information extraction, spam detection and relation analysis to name a few. In these applications, the

instances of FSM can get very large and may include billions of sequences. Ex. Google published a corpus of more than 1 billion n-grams. Hence, distributed and scalable FSM algorithms are a necessity. MG-FSM is a scalable, distributed frequent sequence mining algorithm that can support gap-constraints. The algorithm partitions and rewrites the set of input sequences in a way that each partition can be mined independently and in parallel. No post-processing of results across partitions is needed. MG-FSM makes use of a maximal gap parameter, γ, to find combination of words in a sequence in proximity. In n-gram mining, $\gamma = 0$ and in case of unconstrained FSM, γ = infinity.

A sequence database $D = \{S_1, S_2, \dots, S_D\}$ is a multiset of input sequences. A sequence is an ordered list of items from some dictionary $\sum = \{w_1,\dots,w_{|\sum|}\}$. We write $S = s_1, s_2, \dots, s_{|S|}$ to denote a sequence of length $|S|$ where $s_i \in \sum$ for $1 \leq i \leq |S|$.

$\sum^+$ denotes a set of all non-empty sequences constructed with items from $\sum$. If T is an input sequence in the database, S an arbitrary sequence and the maximum gap parameter $\gamma \geq 0$ then we say, S is a γ-subsequence of T denoted by $S \subseteq \gamma\ T$, with a gap of utmost γ between consecutive items. Standard n-grams are 0-subsequences. Ex. If T = abcd, S_1 = acd and S_2 = bc, then $S_1 \subseteq_1 T$ (but $S_1 \not\subseteq_0 T$) and $S_2 \subseteq_0 T$.

The γ – support $Sup_\gamma(S,D) = \{T \in D : S \subseteq \gamma\ T\}$. Frequency of sequence S is $f_\gamma(S,D)$. Support threshold is denoted by σ, for $\sigma > 0$ we say that sequence S is (σ, γ)-frequent if $f_\gamma(S,D) \geq \sigma$.

$F_{\sigma,\gamma,\lambda}(D)$ is a set of all (σ, γ)-frequent sequences in D of length at most λ. Example: if D = {abcaaabc, abcbbabc, abcccabc}, we obtain $F_{3.0.2}(D)$ = {a,b,c,ab,bc}, $F_{3.1.2}(D)$ = {a,b,c,ab,ac,bc} and $F_{3.2.2}(D)$ = {a,b,c,ab,ac,bc,ca}. [13]

Machine Learning Classifiers

This section is a quick overview of a few different machine learning classifiers which will be used in our research.

KNN Classifier

K-Nearest Neighbors is a supervised machine learning model. This model works by taking a data point and looking at its ‘k’ closest labelled data points and assigning the label of most data points to the test point. KNN is one of the simplest forms of machine learning algorithms and is mostly used for classification problems. KNN can be used as a choice of model when the dataset is properly labelled, is noise free and is small in size. The model slows down considerably as the size of the dataset increases. As a distance metric, mostly Euclidean distance is used. The advantage of this model is that it is simple, and the k value is the only hyperparameter that needs to be optimized. We used GridSearchCV to find the best k value for our classifier using the different datasets. [14]

Decision Tree Classifier

Decision Tree is another type of supervised learning algorithm mainly used in classification problems. In classification, decision tree only checks one feature at a time. Advantages of decision tree are that it is fast during the training and testing stages and it is easy to understand. A top node called the root node is available at the top of the tree. The splitting process comes to an end when there are no more nodes to split. The node at the bottom of the tree will determine the class for the condition and is referred to as the leaf node. [15] We use GridSearchCV to tune the hyperparameters.

Random Forest Classifier

Random Forest is a flexible and effective algorithm that even produces great results without hyperparameter tuning. This algorithm builds multiple decision trees and merges them together to get a more accurate and stable prediction. Random Forest has almost similar hyperparameters as that of Decision Tree. Instead of finding the most important feature while splitting the node, the classifier uses the best feature among the subset of features generally resulting in a better model. Random Forest is extremely versatile and is well suited for both classification and regression problems. The classifier also prevents overfitting provided there are enough trees in the forest. The main limitation is that as the number of trees in the forest increase, the algorithm can become significantly slower. [16]

AdaBoost Classifier

AdaBoost or Adaptive Boosting is a supervised learning technique used in Machine Learning. It is called Adaptive Boosting as the weights are re-assigned for each instance with higher weights given to incorrect classifications. The algorithm initially makes n decision trees during the training period. After the first decision, incorrect classifications are given more priority and only these records are sent as inputs to the second model. The number of repetitions can be defined in the hyperparameters. In this classifier, the algorithm initially only makes one node with two leaves known as stump. The error of the first stump influences how the other stump is made. The output is decided based on most of the total number of stumps. [17]

Support Vector Machine Classifier

In Support Vector Machine algorithm, each data item is plotted in a n-dimensional space where n is the number of features. The classes are segregated using a hyper-plane. A hyper-plane is selected based on how well it segregates the classes. Since real-time data is never completely linear, SVM addresses non-linearity by using soft margins and kernel tricks. Soft margin is a concept using which the classifier tolerates a few incorrectly classified data points. Using kernel tricks, it tries to find a non-linear decision boundary. This algorithm works well when there exists a clear margin of separation. However, this classifier is not suited for very large datasets. [18]

Classifier Performance Evaluation Metrics

Metrics like True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN) were used to evaluate the performance of our method. TP is the number of malware samples that are correctly predicted, TN is the number of goodware that are correctly predicted, FP is the number of goodware samples predicted as malware and FN is the number of malware samples predicted as goodware.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F - measure = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Matthews Correlation Coefficient (MCC) is also reported for the classifier. MCC provides a measurement of the quality of binary classifications. It has a value between -1 and 1, where 1 indicates perfect prediction, -1 is inverse prediction and 0 means it does not work better than a random prediction.

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(FP + TP)(FN + TP)(FP + TN)(TN + FN)}}$$

Receiver Operating Characteristics (ROC) and Area Under Curve (AUC) were also reported for every classifier. ROC curves plot TP rate on the Y-axis vs FP rate on the X-axis signifying the top left corner of the plot to be ideal with a FP rate of 0 and TP rate of 1. The larger the AUC, the better the classifier is at distinguishing between classes. We run 10 times 10-fold cross validation on all the classifiers and plot the ROC curves to inspect the performance.

RECENT WORKS

Malware detection is an on-going research topic and there are various works on using different static and dynamic approaches. Santos et al, [1], proposed a method to classify malware variants by constructing a vector representation of the executables based on the length of opcodes. Schultz et al, [19], proposed using non-overlapping sequence of bytes of a given length n as features to train machine learning classifiers. Brezo et al, [20], proposed another static malware analysis approach where mutual information of opcodes was used based on its frequency to train the machine learning classifiers. Darabian at al, [3], proposed the MSP approach which is the basis of our research. Classifiers were evaluated not just for unseen dataset but also for polymorphed datasets. Azmoodeh et al, [21], proposed an interesting method of malware detection in IoT networks based on energy consumption patterns. This method was specifically proposed for detecting ransomware attacks by monitoring power consumption of Android devices. The unique local fingerprint of ransomware's energy consumption was used to distinguish between malicious and benign applications.

Another work by Azmoodeh et al, [22], focused on detecting malware but in Internet of Battlefield Things. The proposed method uses class-wise selection of opcode sequences as a feature for classification. Deep Eigenspace learning approach was used for malware classification. An opcode based approach was introduced by HaddadPajoug et

al, [23], which explores the potential of Recurrent Neural Network deep learning in detecting IoT malware. Features were selected based on the distributed frequency of opcodes in both malware and benign classes using Information Gain. An interesting approach was introduced by Su et al, [24], to classify IoT based malware using image recognition. This is a lightweight scheme to detect IoT DDoS malware on local IoT devices. The detector is based on convolutional neural networks and can be tuned more to improve efficiency and reduce network size.

As seen there have been a lot of work proving the utility of opcodes in the application of malware detection. Homayoun et al, [25], used sequential pattern mining techniques to find maximal frequent patterns within various ransomware DLL, Filesystem and Registry activities. Shabtai et al, [26], provided a taxonomy for malware detection using machine learning algorithms by reporting various feature selection techniques like gain ratio, fisher score, document frequency, and hierarchical feature selection. Different classification algorithms like Artificial Neural Networks, Bayesian Networks, Naïve Bayes, K-nearest neighbors among others were used.

In our work, we use the previous work as the basis and propose a malware detection approach using maximal frequent sequences

PROPOSED APPROACH

This section illustrates the proposed approach and the steps for implementation. In the first subsection, we discuss how the dataset was created and then three approaches are introduced to distinguish between malware and benign opcodes.

Dataset

A dataset of IoT malware and benign samples is used. Most IoT platforms are ARM-based, hence 32-bit ARM-based malware was collected to depict real world attacks applicable to the IoT environment.

Dataset Collection

ARM-based malware samples were collected from VirusShare. [27] Benign samples were collected from Linux Debian package repositories. [28]

Dataset Preprocessing

All samples are disassembled using IDA Pro to extract opcodes. [29] We create sequential datasets in the format $D = \{S_1, S_2, \dots, S_n\}$ where n is the total number of sequences and S_i is a sequentially ordered set of opcodes .

Four datasets were created for goodware and malware samples. $D_{goodware}$ refers to the benign samples whereas $D_{malware}$ refers to the malware samples. We collected 244 malware samples and 268 benign samples. 15% of $D_{malware}$ and $D_{goodware}$ is randomly

separated to be used as unseen data for the testing stage. These form $D_{UnseenGoodware}$ and $D_{Unseenmalware}$.

Table 1 Dataset Collected

Dataset	**Number of sequences**
$D_{malware}$	207
$D_{goodware}$	228
$D_{UnseenMalware}$	37
$D_{UnseenGoodware}$	40

To detect polymorphed malware, a polymorphed version of the unseen malware dataset is created. The polymorph code replaces an opcode with functionally equivalent opcode wherein λ determines the percentage of opcodes to be polymorphed. Table 2 illustrates this. To determine a polymorphed version, first a set of rules for obfuscation were established. The opcodes were morphed based on the percentage of obfuscation defined which satisfy the rules. Six polymorphed versions of the unseen malware datasets were created which were used to test the viability of the approaches in detection of polymorphed malware.

Table 2 Polymorphed Malware Dataset

λ (between 0 and 1)	Dataset name
0	$D_{morphed0\%}$
0.2	$D_{morphed20\%}$
0.4	$D_{morphed40\%}$
0.6	$D_{morphed60\%}$
0.8	$D_{morphed80\%}$
1	$D_{morphed100\%}$

Opcode Rank Approach

Methodology

Some opcodes have a higher frequency than other in malware. Hence all the opcodes in the $D_{malware}$ dataset are counted and a dictionary called opcode_rank_dict() = {opcode: rank} is created where the opcode with the highest frequency has rank 1 and so on. Table 3 illustrates the top 5 opcodes and their frequencies in our malware dataset.

Table 3 Opcode Frequency Rank: Top 5

Opcode	Frequency	Rank
mov	661495	1
ldr	483457	2
add	210315	3
str	199028	4
cmp	185033	5

To run machine learning classifiers, vector format of the opcodes is needed. The malware opcodes are substituted with their opcode ranks based on the opcode_rank_dict to perform vectorization. Example: [mov ldr add] → [1 2 3]. This replacement process is also done on the goodware samples to generate vectors using the same dictionary. These numerical sequences are then used to train the classifiers and test suitability in malware classification. The vectors are equalized using $D_{malware}$ median length which is 15177 in our case. Vectors less than the median are padded with zeroes and more than the median are truncated. The process in demonstrated in Algorithm 1.

Algorithm 1: Pseudocode to create vectored dataset using opcode-rank dictionary

```
text_list ← text_file from malware folder
for line in text_list do
    split line into opcodes
    for opcode in opcodes do
        if opcode in opcode_frequency_dict then increment its value
        else add it to opcode_frequency_dict with value 1
        end if
    end for
end for
rank = 1
sort opcode_frequency_dict based on frequency
for (key,value) pair in opcode_frequency_dict do
    append opcode_rank_dict (key, rank)
    rank ← rank + 1
end for
replace opcodes in mal_dataset with corresponding rank from the
opcode_rank_dict()
calculate median of all vectors in mal_dataset
equalize all vectors in mal_dataset {Replacement based on malware median
done for goodware samples as well}
end
```

Evaluation

Performance of machine learning classifiers like Decision Tree, KNN, Random-Forest, SVM and AdaBoost is evaluated using 10 times 10-fold cross validation which is a repeated k-fold cross validation technique. ROC curves for all the classifiers are reported as shown in Figure 2. The similarities in the curves show that the classifiers behave similarly indicating the suitability of opcodes for the classification.

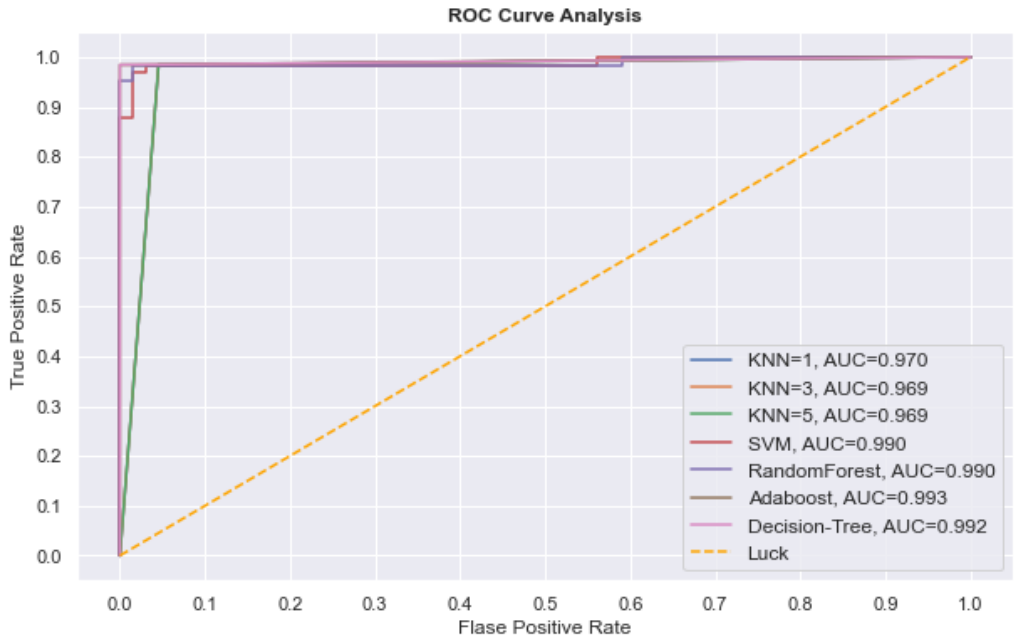

Figure 2 ROC curves - opcode rank

Accuracy and F-measure using 10 times 10-fold cross validation is also measured for all the classifiers as shown in Table 4. We see that all the classifiers perform in a similar manner and the Decision Tree classifier has the best performance. Decision Tree

and Adaboost have MCC values very close to 1 as shown in Figure 3. KNN, the least sophisticated classifier, also has a good MCC of 0.94.

Table 4 Accuracy and F-measure: Opcode Rank

Classifier	Accuracy	F-measure
SVM	97.9	97.6
Random Forest	98.8	96.9
Decision Tree	99.3	99.2
Knn = 1	98.1	97
Knn = 3	97.4	96.9
Knn = 5	96.9	96.2
Adaboost	97.6	97.5

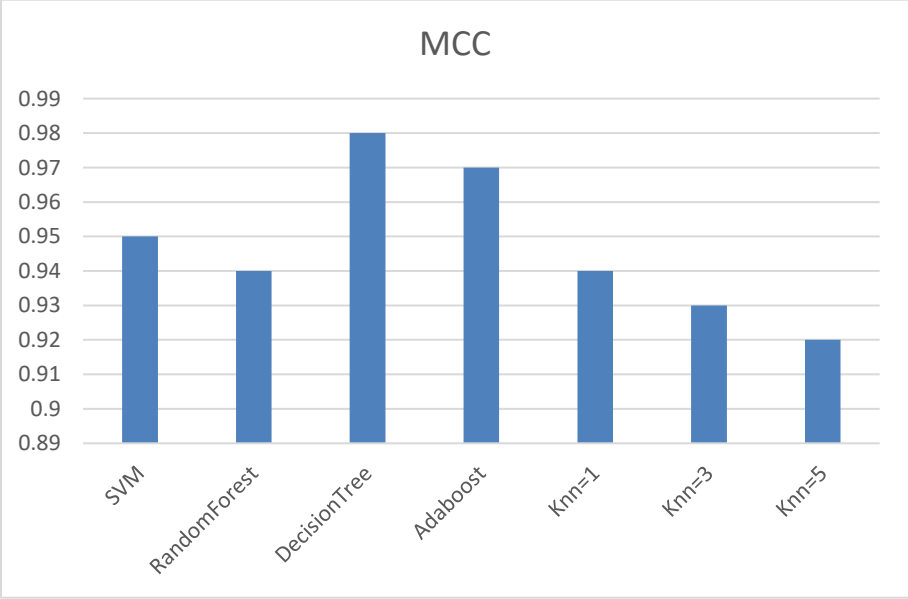

Figure 3 MCC: Opcode Rank

We now evaluated the performance of these classifiers on unseen datasets with no obfuscation. Our classifiers performance is comparable on unseen dataset as well as shown in Table 5. We obtained accuracies of upto 99% using the AdaBoost classifier.

Table 5 Accuracy on Unseen: Opcode Rank

Classifier	**Accuracy**
SVM	96.2
Random Forest	96.5
Decision Tree	98.1
KNN = 1	97.4
KNN = 3	96.4
KNN = 5	94.2
Adaboost	99.1

We then tested this approach on polymorphed dataset to evaluate its viability. We observed that the accuracy of Decision Tree dropped to about 55% and KNN has the best values. The MCC values of Adaboost and Decision Tree is close to 0 as the percentage of polymorphism increases indicating that their performance is almost random which can be seen in Figure 5. KNN has the best performance in this approach but has the disadvantage of being slow.

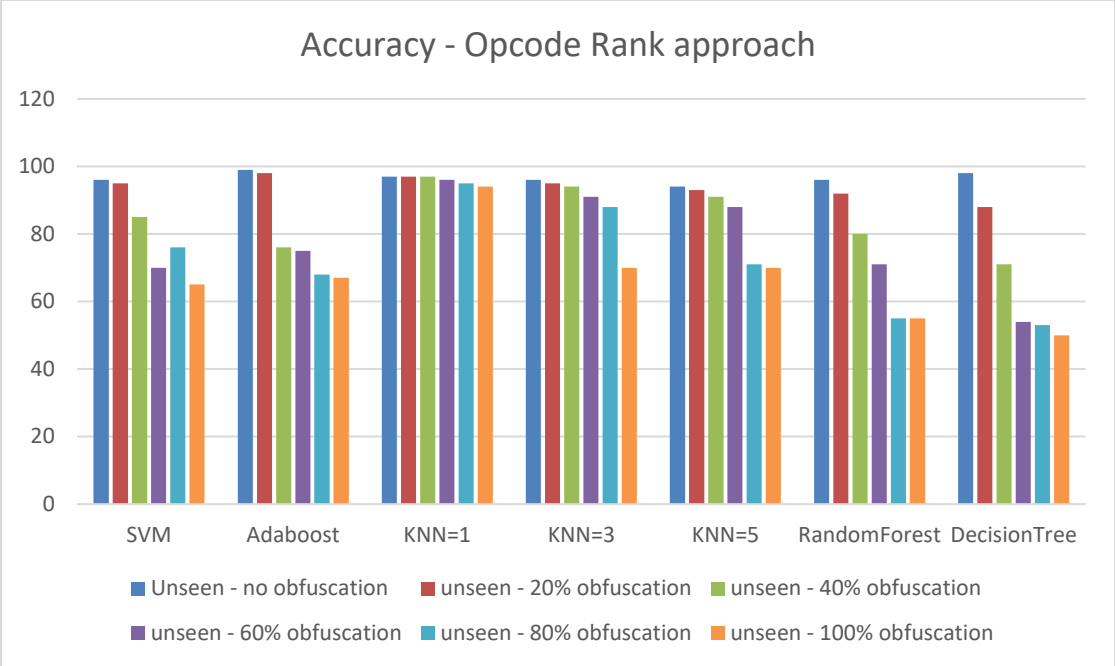

Figure 4 Accuracy on polymorphed: Opcode Rank

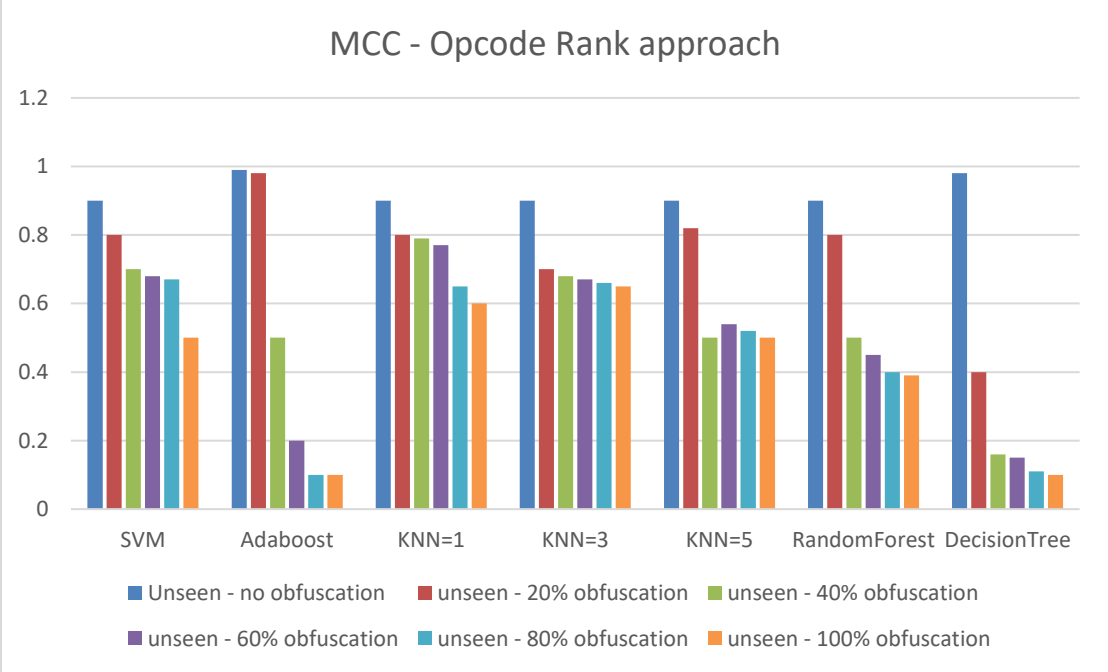

Figure 5 MCC on polymorphed: Opcode Rank

Sequential Pattern Mining Approach

We used MG-FSM to mine Maximal Sequential Patterns (MSPs) for classification. If a sequence does not have a super-sequence, then it is called MSP. The collection of all MSPs of the dataset is denoted by Maximal Sequential Pattern Collection, MC_D. The elements in the MC_D are in the format (MSP, sup_{MSP}) where sup_{MSP} is the numbers sequences the MSP occurs in the dataset. We run MG-FSM on the malware dataset to mine the MSPs.

Parameters used to mine the MSPs in our research were:

1. Gap (γ): maximum gap parameter, $\gamma = 0$
2. Support threshold (σ_{min}): minimum number of sequences the subsequence is present in. $\sigma_{min} = 103$ (50% of malware dataset)
3. Length (λ): maximum length of the subsequence, $\lambda = 5$
4. Type (m): Maximal type is chosen to only pick MSPs

Number of MSPs collected in $MC_D = 5742$.

MSP list method

We now build a new vectored dataset using the mined MSPs to run machine learning classifiers. The frequency of each MSP from MC_D is counted for every sequence of the given dataset. These frequency values corresponding to the MSPs are used to build the vectored dataset. The (n x m) dataset is created where, n = number of sequences and m = 5742. The number of columns has now reduced from 15177 in the opcode approach to 5742. This new dataset is then used to evaluate the classifiers and their performance.

Algorithm 2: Pseudocode to create vectored dataset using MSP list method

```
final_dataset = {empty list}
for each_sequence in malware_folder do
    seq_list ← each_sequence
    for single_msp in MC_D do
        feature_vector = {empty list}
        calculate_occurrence(single_msp, seq_list) {calculates frequency in rep}
        append feature_vector with rep {repeat for every MSP}
    end for
append final_dataset with feature_vector
end for
```

Evaluation – MSP list

We first report ROC-AUC curves and as seen in Figure 6, the similarity in ROC curves indicate the viability of using MSPs for classification.

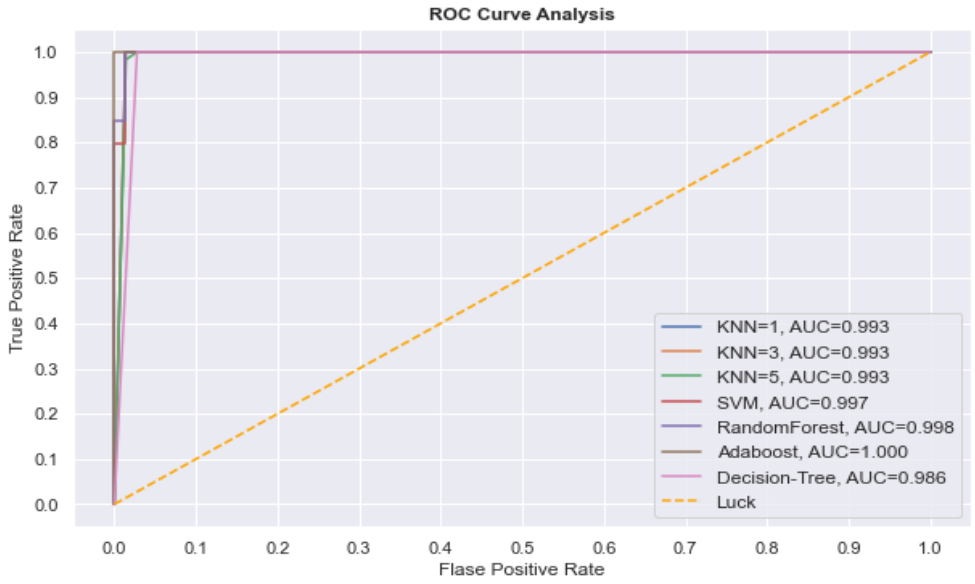

Figure 6 ROC curves: MSP list

Accuracy and f-measure using 10 times 10-fold cross validation is reported and it is seen that the classifiers have comparable performance. As seen in Table 6, Random Forest and Adaboost outperform all the other classifiers. However, Random Forest has the least MCC when compared to others.

Table 6 Accuracy and F-measure: MSP list

Classifier	Accuracy	F-measure
SVM	97.5	96.8
Random Forest	99.8	99.2
Decision Tree	99.3	99.2
Knn = 1	99.5	98.5
Knn = 3	99.4	98.4
Knn = 5	99.1	97.7
Adaboost	99.7	99.2

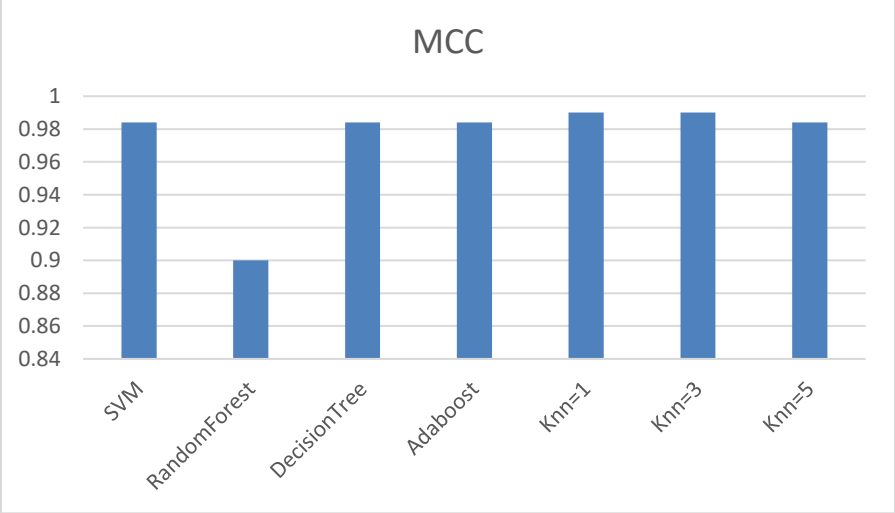

Figure 7 MCC: MSP list

The approach was then tested on unseen data and accuracy was measured. All the classifiers apart from Random Forest had comparable values.

Table 7 Accuracy on Unseen: MSP list

Classifier	Accuracy
SVM	99
Random Forest	94
Decision Tree	99.1
KNN = 1	99.6
KNN = 3	99.7
KNN = 5	99.5
Adaboost	99.5

Finally, the accuracy and MCC values were reported for polymorphed unseen malware. AdaBoost had the highest accuracy and MCC values proving its suitability for this approach. As the percentage of polymorphism increases, it is seen that the MCC values reduce considerably especially for KNN classifier.

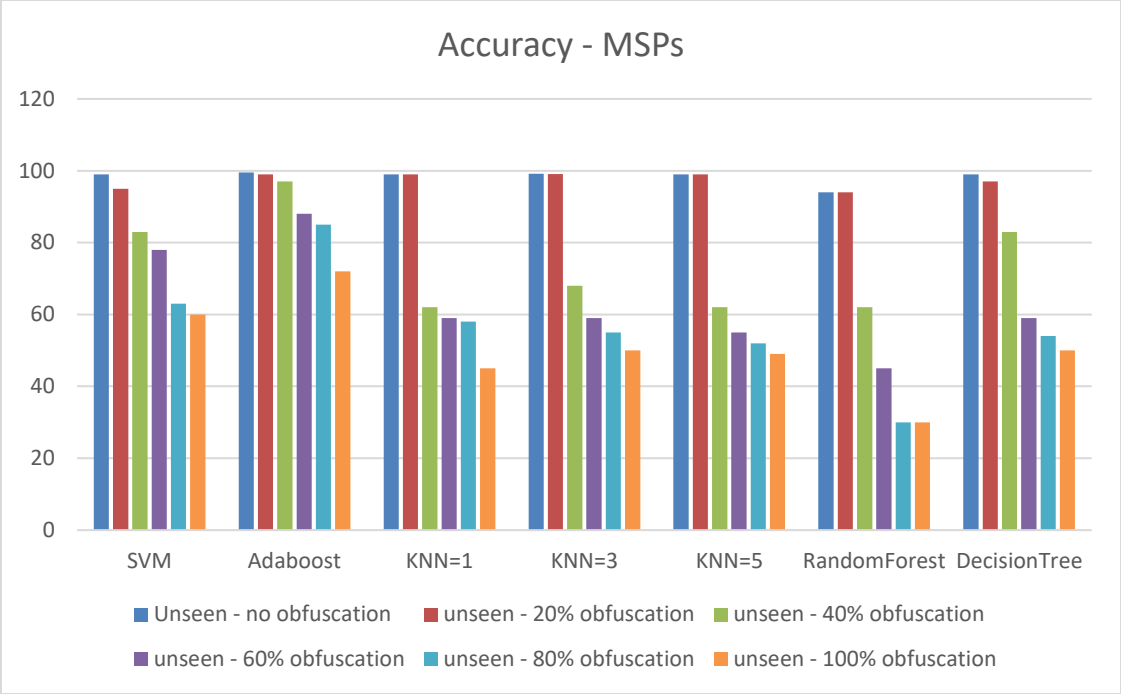

Figure 8 Accuracy on polymorphed: MSP list

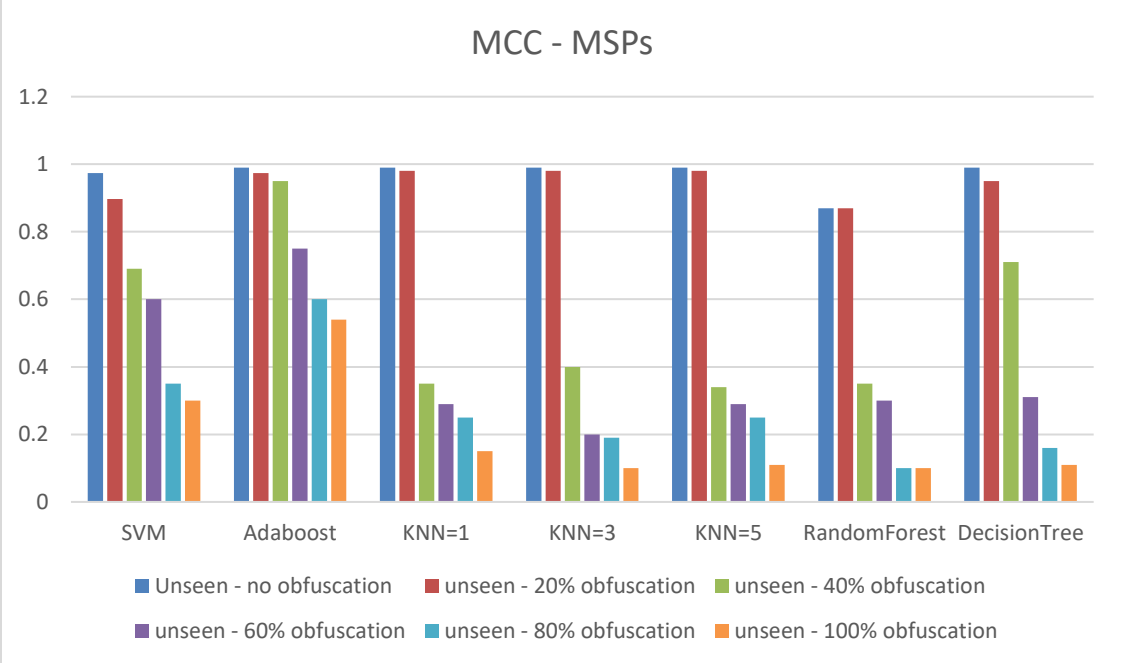

Figure 9 MCC on polymorphed: MSP list

MSP type method

In this method, we extract features based on the MSP type. Before creating a feature dataset, first support values for each MSP in MC_D is calculated. The support ratio (SR) of MSP is the possibility of occurrence of the MSP in malware dataset. The SR value is calculated for each MSP and is stored in a new dictionary called msp_sr_dict = {MSP: SR_{MSP}}. This SR value is used further into feature extraction.

$$SR = \frac{sup_{MSP}}{total\ malware\ sequences}$$

The opcodes are divided into categories based on their functionality by referring to the ARM resources. This is demonstrated in Table 8.

Table 8 Opcode Categorization

Opcodes	Category	Functionality
AND, EOR, SUB, RSB, ADD, ADC, SBC, RSC, TST, TEQ, CMP, CMN, ORR, MOV, BIC, MVN, CDP	C_1	Common addition, subtraction, and, or
MUL, MULL, MLA, MLAL	C_2	Multiplication
B, BL, BX	C_3	Branch
LDM, STM, LDR, STR, LDC, STC, MRC, MCR	C_4	Load/Store
MRS, MSR	C_5	Coprocessor registers
DMB, DSB, ISB, NOP, SEV, SVC, WFE, WFI, YIELD, SWI, SWP, ADR, FLDM	C_6	Rarely used/miscellaneous

The MSPs from MC_D can be divided into two main types based on the categories: atomic and transitional. Atomic MSPs are the ones whose opcodes all belong to one

category while the opcodes of a transitional MSP belong to more than one category. Example: MSP1 = {B, BL, BX} is atomic because all opcodes belong to C_3. MSP2 = {BL, BL, ADD, MOV} is transitional because {BL, BL} belongs to C_3 whereas {ADD, MOV} belongs to C_1.

Based on our categorization, the number of atomic MSPs are 6. The equation to calculate the total number atomic and transitional MSPs based on the number of transitions, x, is:

$$Total\_MSP\_types = 6 + \sum_{i=1}^{x} 6 \times 5^i$$

In our study, to keep the number of features less, we only consider the first transition, hence x = 1. The total number of MSP types in our case is 36. The number of MSP types are used as features for classification; hence each sequence now has 36 features associated with it. Since we are only working with the first transition, we denote them as $C_{i\text{-}j}$ where i is the number of the first category and j is the number of the second category. Table 9 shows some examples of naming the MSP based on its type.

Table 9 MSP type naming convention

MSP	**Type**
B, BX	C_3
AND, SUB, NOP	$C_{1\text{-}6}$
STR, MSR	$C_{4\text{-}5}$
MUL, MLA, CMP	$C_{2\text{-}1}$

Obtaining the MSP type is step one of creating our feature dataset and is demonstrated in Algorithm 3.

Algorithm 3: Pseudocode to obtain MSP type

Given MSP from MC_D

msp_type(MSP**)** {determines MSP type based on type of opcodes}

    msp_type_list = [category of each opcode in MSP] {use opcode category dictionary to determine this list}

    if all elements in *msp_type_list* belong to same category **then** {atomic MSP}
        return category

    else if category(*msp_type_list* [m]) $\in C_i$ & category(*msp_type_list* [n]) $\in C_j$
        return C_{i-j}
    {where $0 \leq m \leq$ len(*msp_type_list* -2) & $m < n \leq$ len(*msp_type_list* -1)}

    end if

End

We create a new (n x m) dataset where n is the number of sequences and m is the number of features, i.e., 36. For each MSP in MC_D, we compute the frequency of the MSP in the sequence, multiply that occurrence with the associated SR value and store the result in the appropriate MSP type feature element. This process is repeated until all the MSPs in MC_D have been checked against the first sequence of the dataset. Finally, this new feature list of 36 elements is appended to the final dataset. We do this for all the sequences and the resulting (n x m) matrix is created as expected. Algorithm 4 demonstrates this process.

Algorithm 4: Pseudocode to create vectored dataset using MSP type

```
final_dataset = {empty matrix}
Given dataset: d
for sequence in d do
    feature_vector = [empty]
    for MSP in MC_D do
        feature_type = msp_type(MSP)
        rep = frequency of MSP in sequence
        feature_vector[feature_type] += rep x SR {Support Ratio computed in a
dictonary msp_sr_dict(MSP : SR) where SR = number of sequences MSP found in
malware dataset * total number of sequences in malware}
    append final_dataset with feature_vector
end for
```

Evaluation – MSP type

The ROC-AUC curves are reported for all the classifiers as seen in Figure 10.

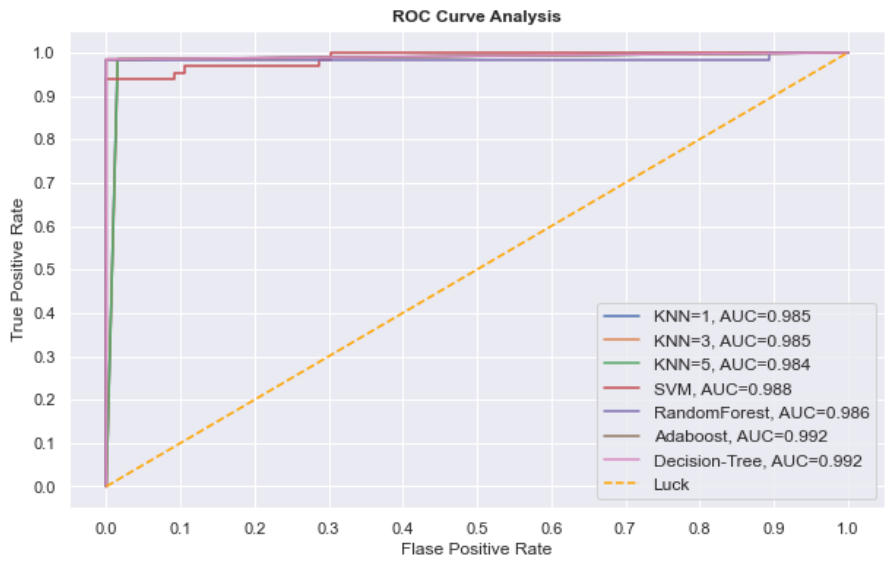

Figure 10 ROC curves: MSP type

The similarities in the curves for the classifiers indicate that MSP type is a useful feature to distinguish between malware and benign samples. Accuracy and f-measure are reported using 10 times 10-fold cross validation. The results are comparable while Random Forest and Adaboost outperform. The MCC values show that the KNN classifier has good performance and is comparable to the other classifiers.

Table 10 Accuracy and F-measure: MSP type

Classifier	Accuracy	F-measure
SVM	97.5	96.8
Random Forest	99.8	99.2
Decision Tree	99.3	99.2
Knn = 1	99.5	98.5
Knn = 3	99.4	98.4
Knn = 5	99.1	97.7
Adaboost	99.7	99.2

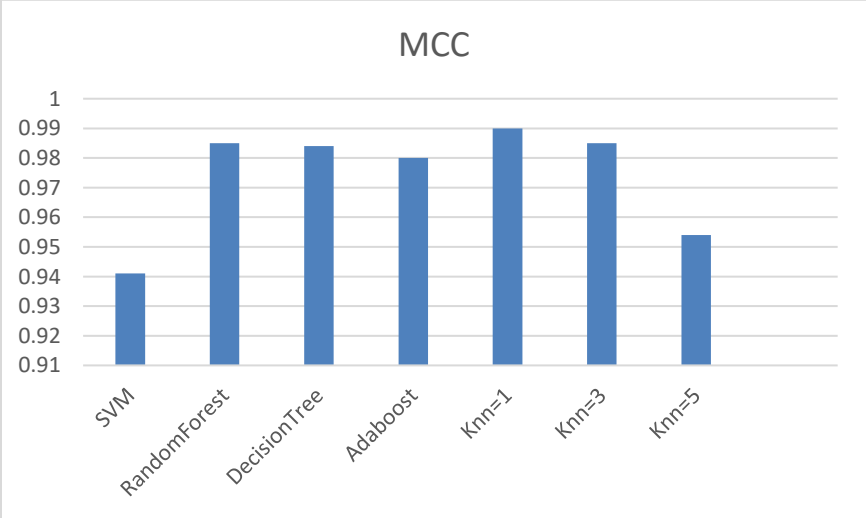

Figure 11 MCC: MSP type

We then compute the accuracy of the trained classifier on the unseen dataset and all classifiers had good performance as seen in Table 11.

Table 11 Accuracy on Unseen: MSP type

Classifier	Accuracy
SVM	99.01
Random Forest	99.21
Decision Tree	98.92
KNN = 1	99.12
KNN = 3	99.54
KNN = 5	97.11
Adaboost	97

This method is tested on polymorphed dataset and we obtained great performance parameters. Accuracy and MCC was reported for all the classifiers on the polymorphed datasets. We see that this method is most suited for detecting polymorphed malware using AdaBoost, Random Forest and Decision Tree Classifiers. KNN has very poor MCC values in this approach and as seen in Figure 13. AdaBoost and Decision Tree outperforms all other classifiers with accuracies close to 99% and MCC close to 1.

In the next section, we compare the performance of all the classifiers using the approaches discussed in this section.

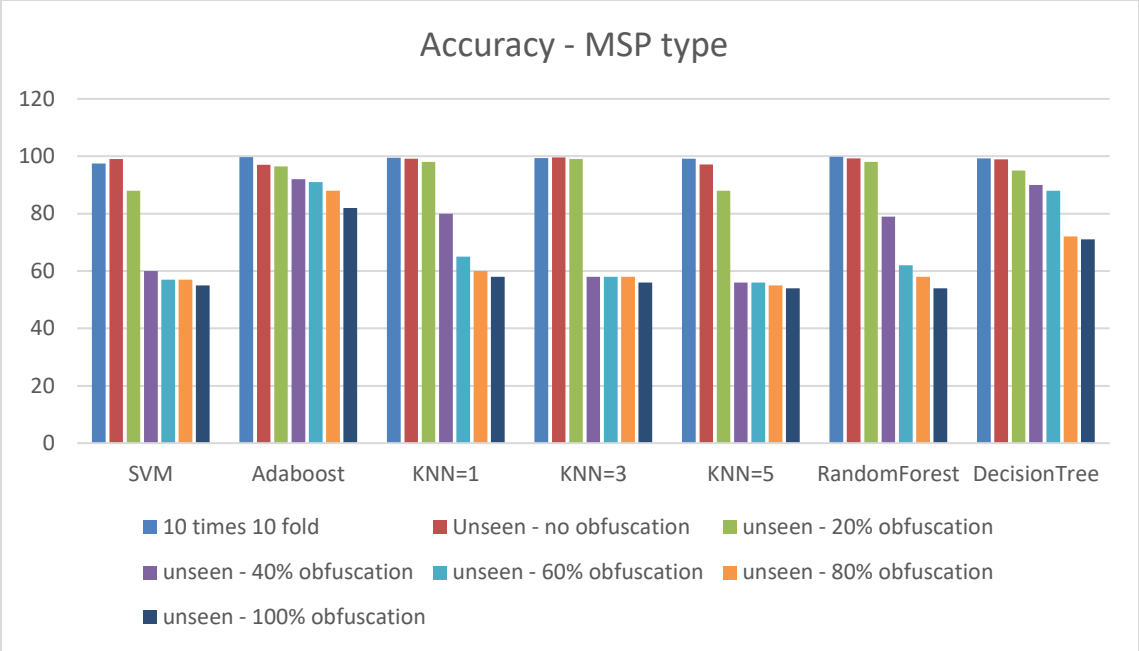

Figure 12 Accuracy on polymorphed: MSP type

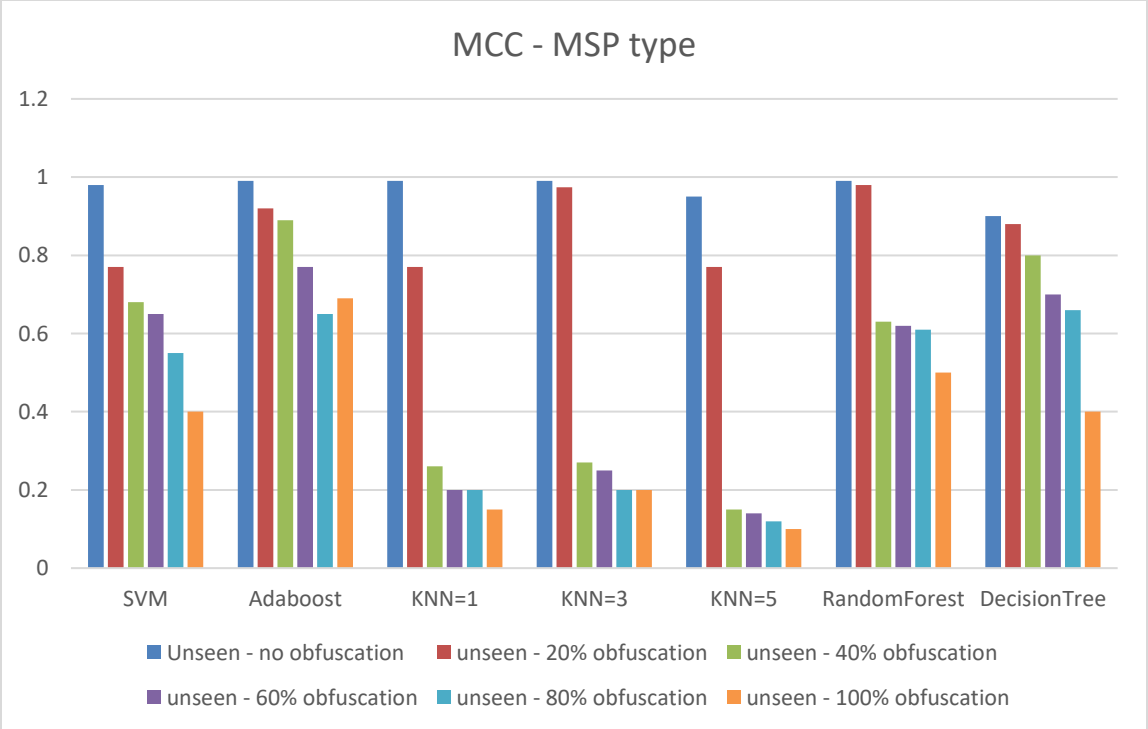

Figure 13 MCC on polymorphed: MSP type

RESULTS

We perform a comparative study of all three approaches on all the classifiers to determine which approach and classifier combination performs the best in classifying polymorphic malware. We can see that all the classifiers have better performance in the SPM method. Both MSP list and MSP type have comparable accuracies over classifiers.

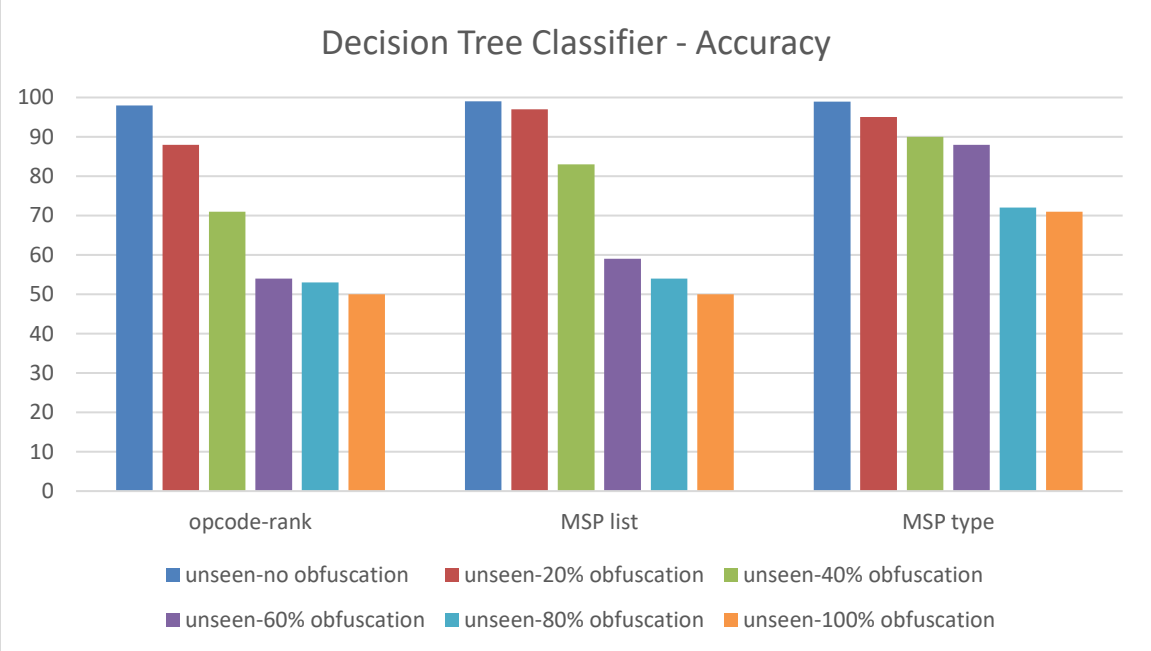

Figure 14 Decision Tree: Accuracy

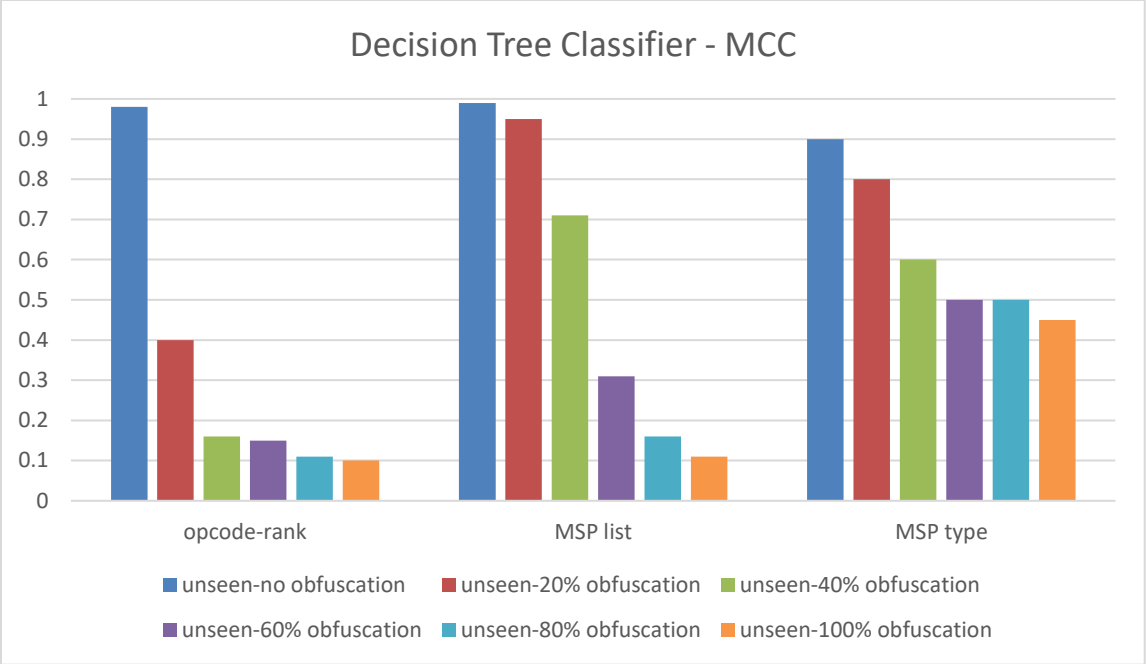

Figure 15 Decision Tree: MCC

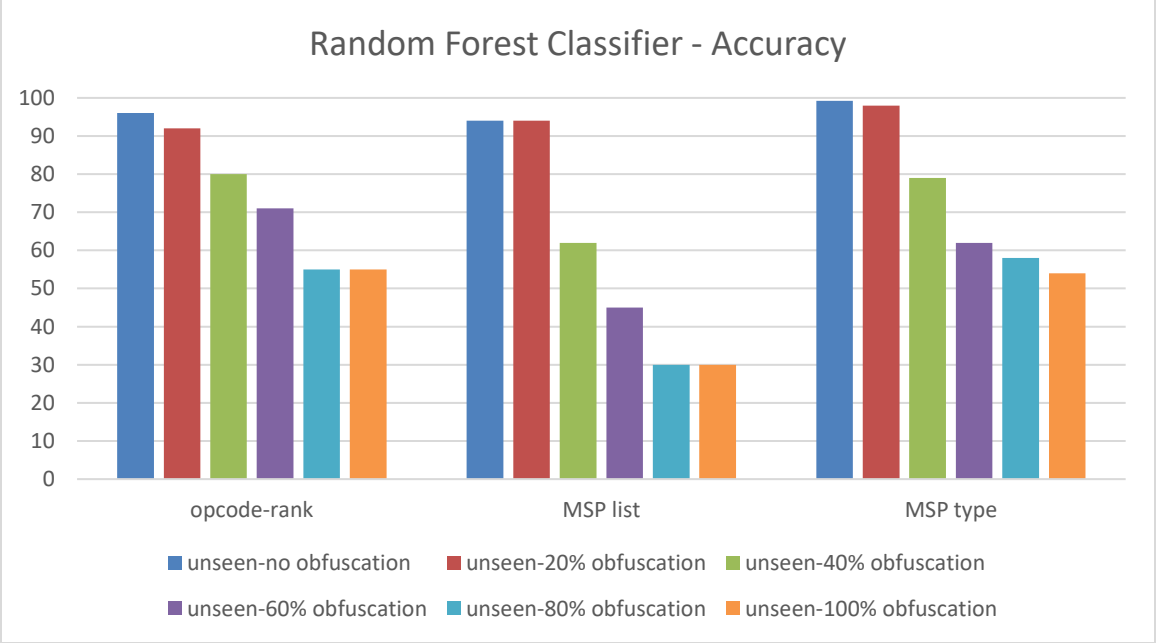

Figure 16 Random Forest: Accuracy

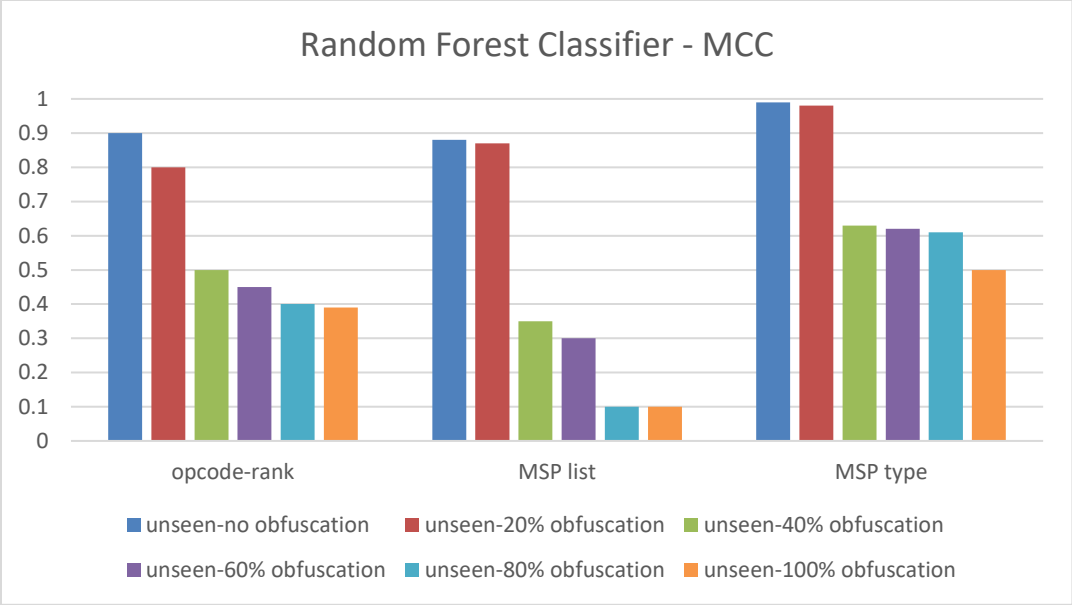

Figure 17 Random Forest: MCC

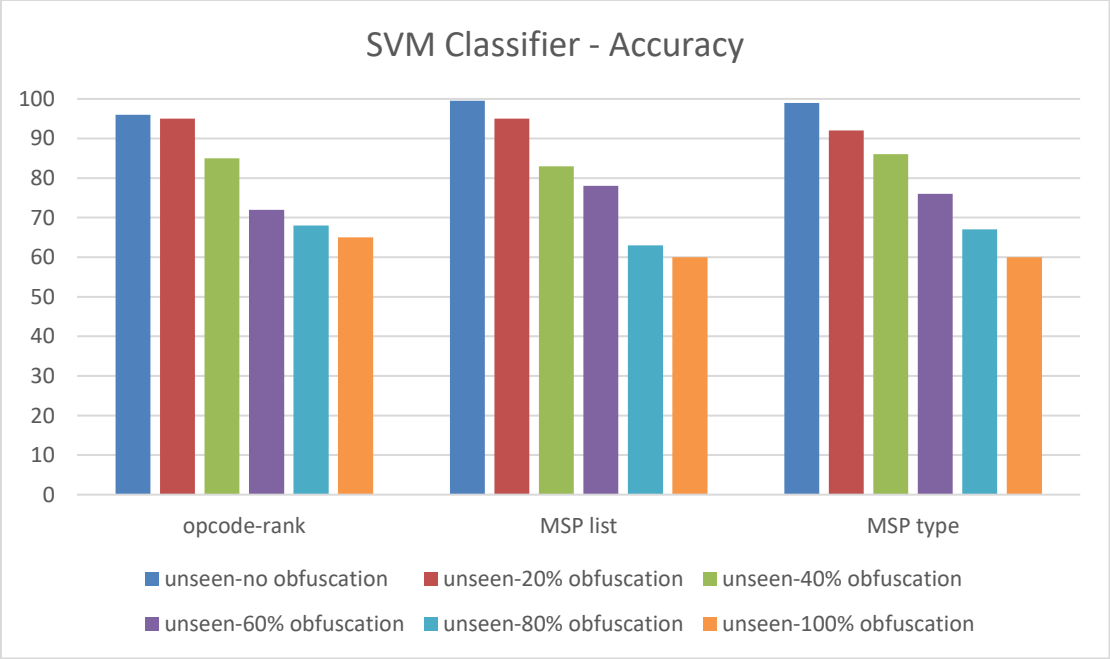

Figure 18 SVM: Accuracy

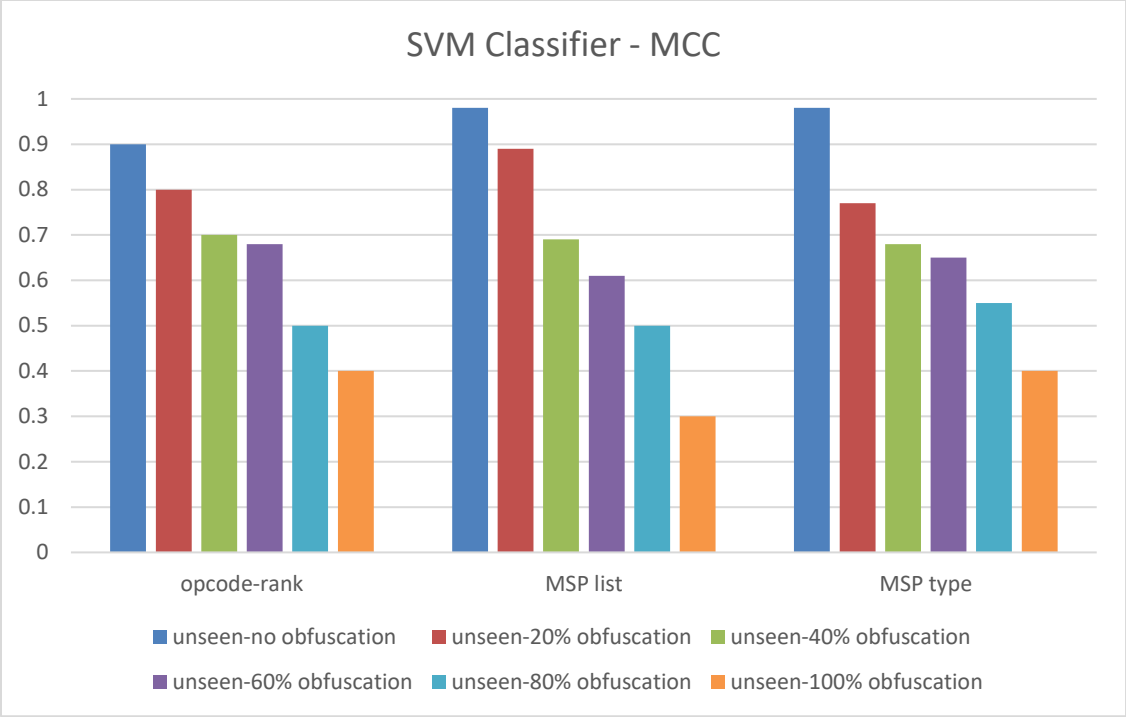

Figure 19 SVM: MCC

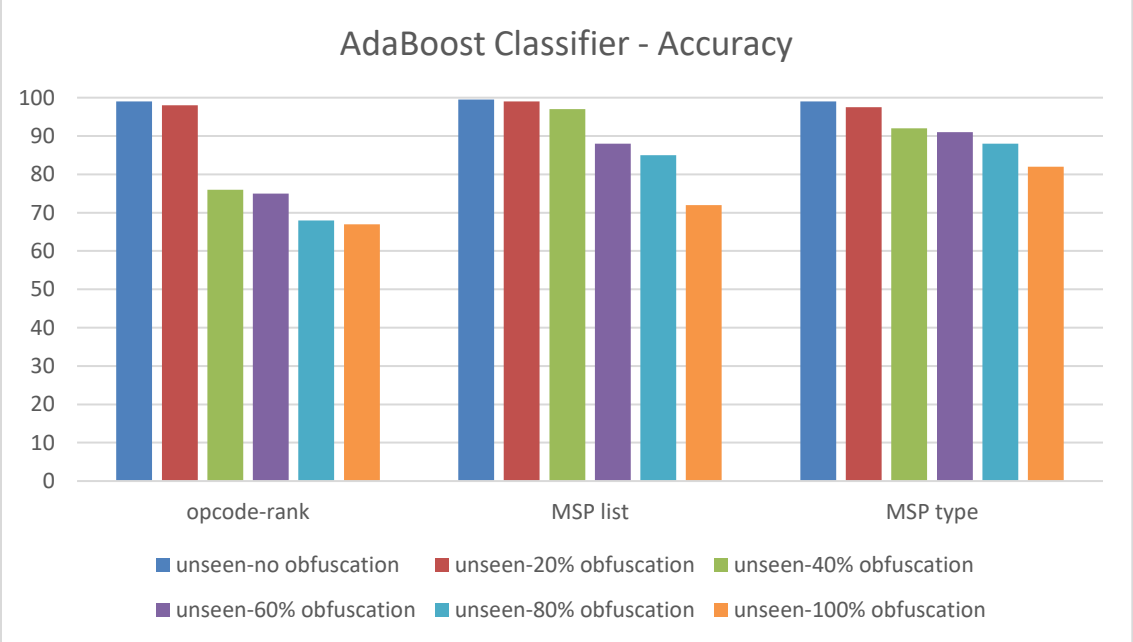

Figure 20 AdaBoost: Accuracy

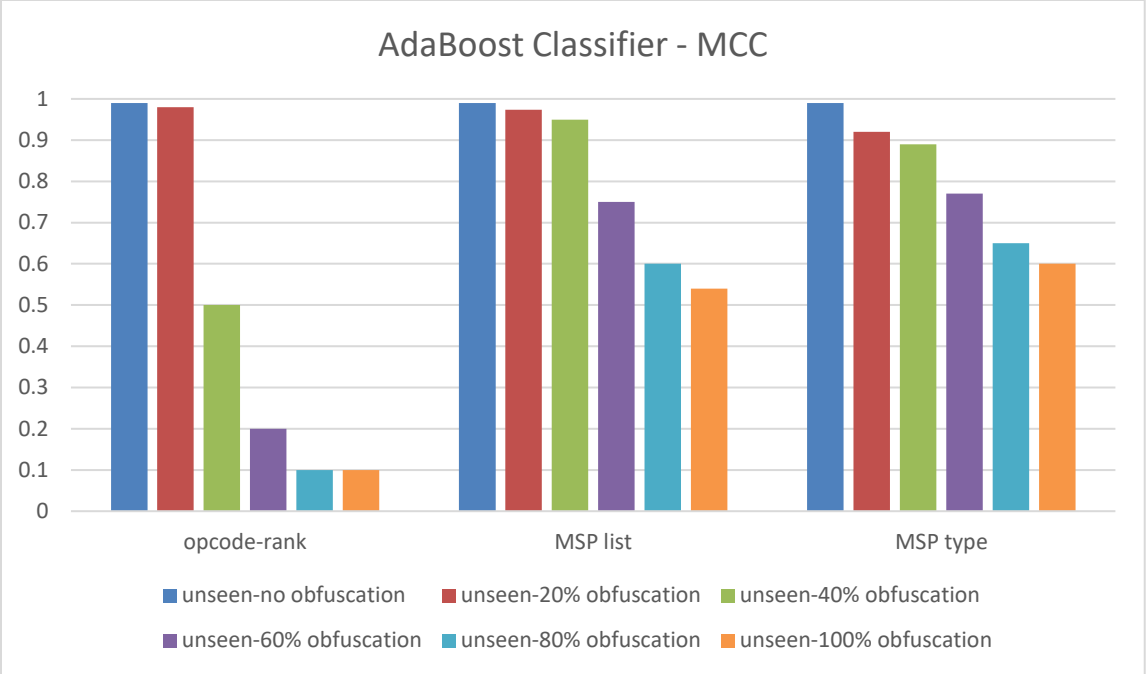

Figure 21 AdaBoost: MCC

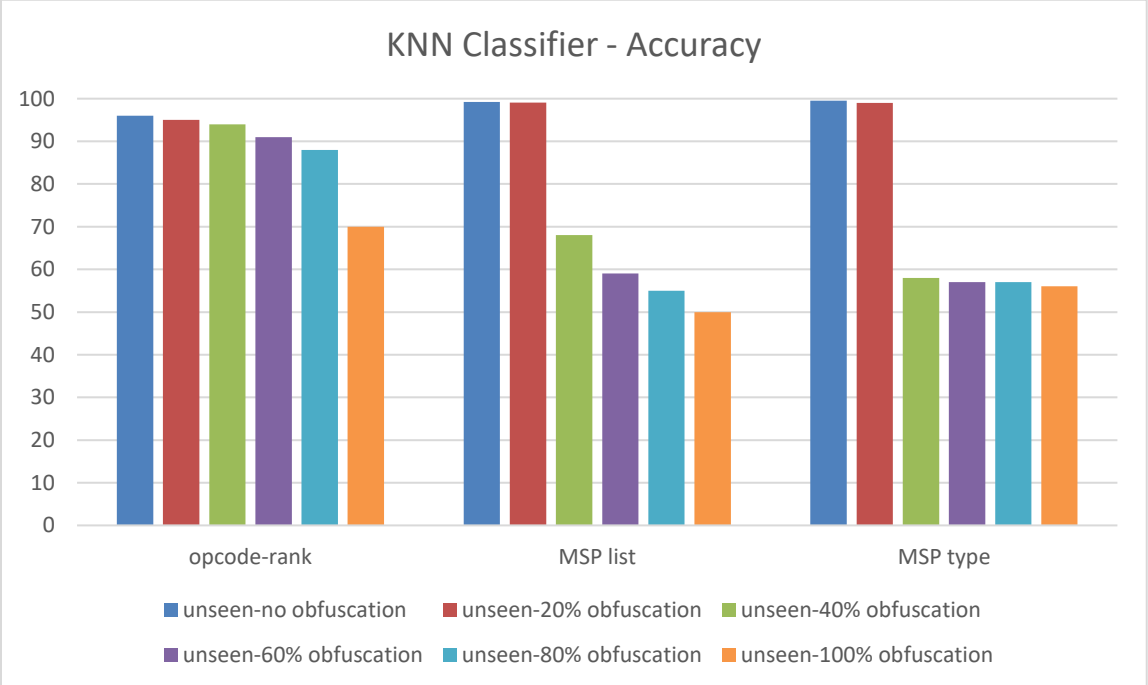

Figure 22 KNN: Accuracy

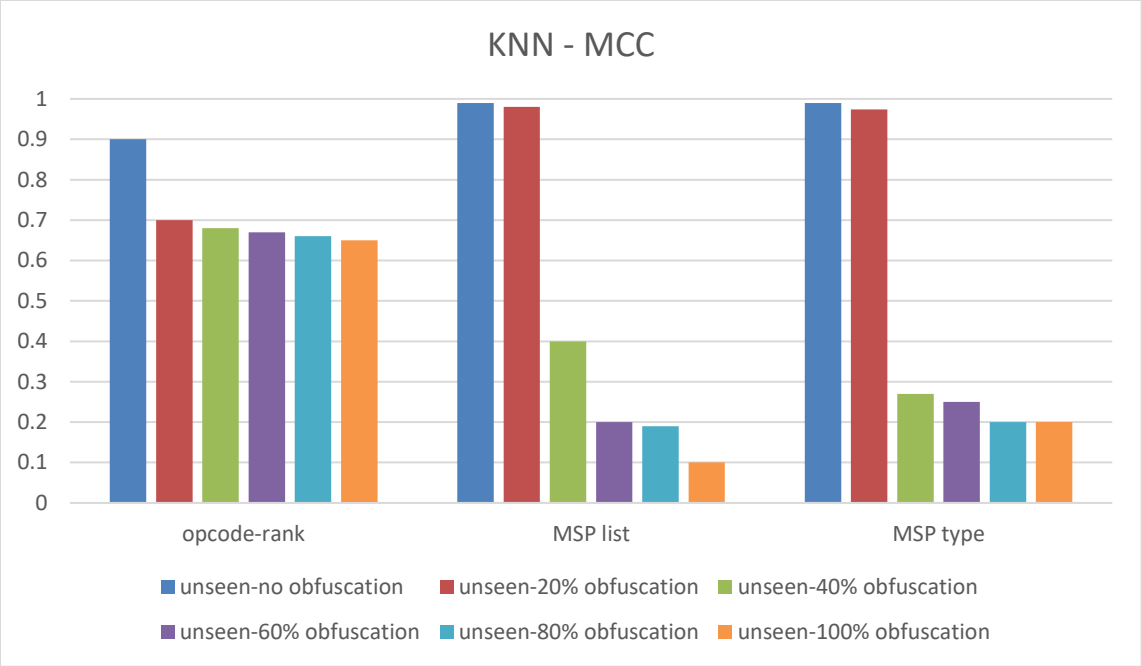

Figure 23 KNN: MCC

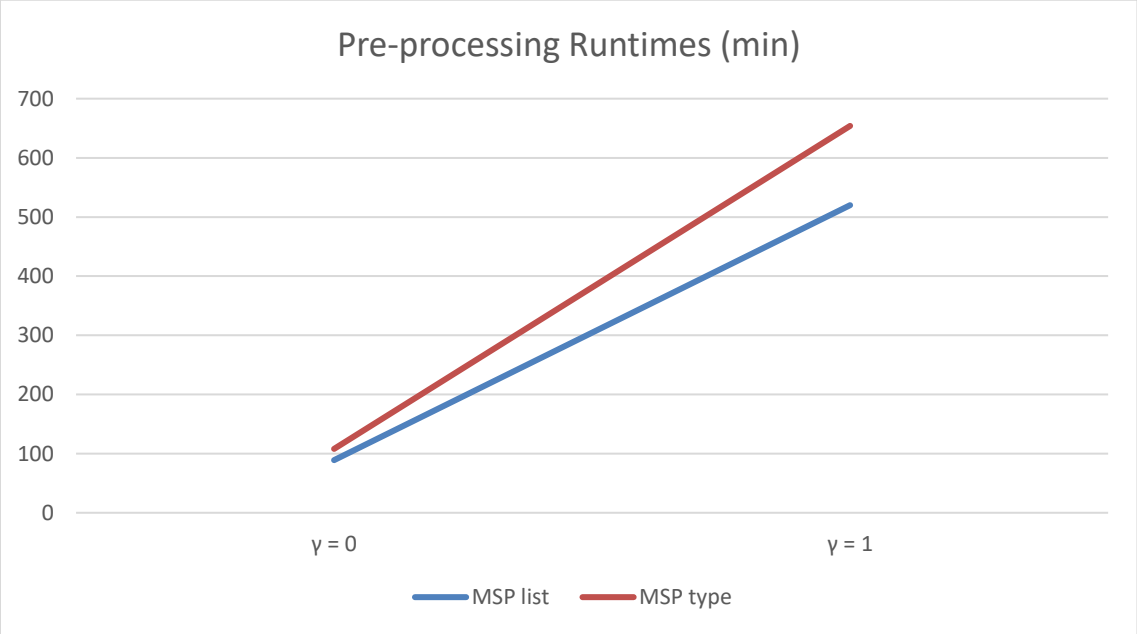

Figure 24 Pre-processing Runtime (mins)

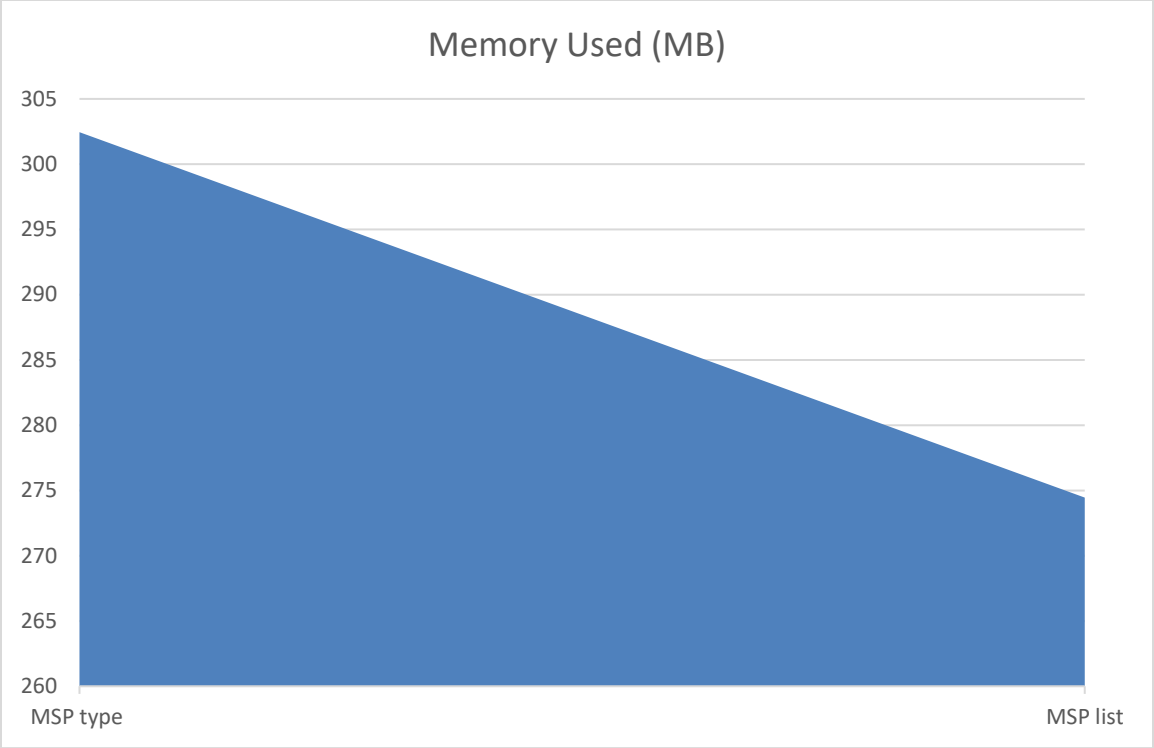

Figure 25 Memory Used (MB)

We also report the pre-processing runtime and memory utilization of the SPM approaches to compare the two. The pre-processing runtime is compared when the gap parameter used to collect MSPs in MC_D is 0 and 1. The number of MSPs in MC_D when gap = 0 is 5742 and when gap = 1 is 34452.

We see that the MSP list approach is not just comparable to the MSP type approach in terms of accuracy but also has faster pre-processing runtime and lower memory usage. The proposed MSP list approach proves its viability in classification of malware.

CONCLUSION

In this work, we propose a static malware detection technique using opcodes. Using MG-FSM, a sequential pattern mining algorithm, we mine the maximal sequential patterns in the malware dataset. The mined MSPs are used for feature extraction in the given sequentially ordered set of opcodes. This work utilized binary classification approach to detect if a given sequence in a dataset is malware or not. The MSPs are used in two different ways to extract features, first, the frequency of MSPs are used and second, the type of MSPs are classified. Both the techniques are tested and compared against an opcode-rank approach wherein the opcodes are ranked based on their frequency in the malware dataset. We observed that all classifiers except KNN, have better performance in the proposed pattern mining approach in detecting polymorphed malware over the opcode-rank. KNN outperforms in the opcode-rank approach since it heavily relies on the distribution of opcodes in the samples. Detecting polymorphed malware is a key element in this research and we see that AdaBoost, Random Forest and Decision Tree Classifiers have the best performance using the MSP methods. The MCC values are more resilient and are closer to 1 in the MSP approach when compared with the opcode-rank as the percentage of polymorphism increases. The proposed MPS list method has comparable performance to the MSP type method even though it has a much simpler implementation. The MSP list approach is faster and has less memory

consumption. The preprocessing time reduces considerably in the MSP list approach even as the number of MSPs increase. The findings demonstrate that our proposed features are a great indicator in malware classification and that the MSP list method is a viable option in classification of malware.

REFERENCES

[1] I. Santos, F. Brezo, J. Nieves, Y. Penya, B. Sanz, C. Laorden and P. Bringas, "Idea: Opcode Sequence Based Malware Detection," in *Porc 2nd International Symposium on Engineering Secure Software and Systems*, 2010.

[2] "IoT Analytics," [Online]. Available: https://iot-analytics.com/. [Accessed 6 November 2020].

[3] H. Darabian, A. Dehghantanha, S. Hashemi, S. Homayoun and K.-K. Choo, "An opcode based technique for polymorphic Internet of Things malware detection," in *Concurrency Computat Prac Exper.*, 2019.

[4] "Security Intelligence," [Online]. Available: https://securityintelligence.com/news/new-kaiji-linux-malware-targeting-iot-devices/. [Accessed 6 November 2020].

[5] I. You and K. Yim, "Malware obfuscation techniques: A brief survey," *International Conference on Broadband, Wireless Computing Communications and Applications,* 2010.

[6] P. O'Kane, S. Sezer and K. McLaughlin, "Obfuscation: The Hidden Malware," *IEEE Security & Privacy,* vol. 9, 2011.

[7] N. Lord, "Digital Guardian," 17 July 2020. [Online]. Available: https://digitalguardian.com/blog/what-polymorphic-malware-definition-and-best-practices-defending-against-polymorphic-malware. [Accessed 26 September 2020].

[8] J. Sowells, "Hacker Combat," 25 April 2019. [Online]. Available: https://hackercombat.com/static-malware-analysis-vs-dynamic-malware-analysis/. [Accessed 22 October 2020].

[9] A. Shalaginov, S. Banin and A. Dehghantanha, "Machine Learning Aided Static Malware Analysis: A Survey and Tutorial," in *Cyber Threat Intelligence,* 2018.

[10] "SAS," [Online]. Available: https://www.sas.com/en_us/insights/analytics/data-mining.html. [Accessed 20 October 2020].

[11] S. Arora, "Simpli Learn," 25 September 2020. [Online]. Available: https://www.simplilearn.com/data-mining-vs-machine-learning-article. [Accessed 15 November 2020].

[12] "Talend," [Online]. Available: https://www.talend.com/resources/data-mining-techniques/. [Accessed 25 October 2020].

[13] I. Miliaraki, K. Berberich, R. Gemulla and S. Zoupanos, "Mind the gap: large-scale

frequent sequence mining," in *ACM SIGMOD International Conference on Management of Data (SIGMOD'13)*, New York, NY, USA, 2013.

[14] E. Allibhai, "Towards Data Science," 26 September 2018. [Online]. Available: https://towardsdatascience.com/buildingaknearestneighborsknnmodelwithscikitlearn. [Accessed 10 September 2020].

[15] H. A. Ismail, "Medium," 11 January 2017. [Online]. Available: https://medium.com/@haydar_ai/. [Accessed 8 September 2020].

[16] N. Donges, "Builtin," 3 September 2020. [Online]. Available: https://builtin.com/data-science/random-forest-algorithm. [Accessed 28 October 2020].

[17] G. L. Team, "Great Learning Blog," 28 May 2020. [Online]. Available: https://www.mygreatlearning.com/blog/adaboost-algorithm/. [Accessed 20 August 2020].

[18] L. Chen, "Towards Data Science," 7 January 2019. [Online]. Available: https://towardsdatascience.com/support-vector-machine-simply-explained-fee28eba5496. [Accessed 15 August 2020].

[19] M. Schultz, E. Zadok and S. Stolfo, "Data Mining Methods for Detection of New Malicious Executables," in *IEEE Symposium on Security and Privacy*, 2001.

[20] I. Santos, F. Brezi, X. Ugarte-Pedrero and P. Bringas, "Opcode sequences as representation of executables," *Information Sciences*, 2013.

[21] A. Azmoodeh, A. Dehghantanha, M. Conti and K. Choo, "Detecting crypto-ransomware in IoT networks based on energy," *J Ambient Intell Human Comput*, 2017.

[22] A. Azmoodeh and A. Dehghantanha, "Robust Malware Detection for Internet of Battlefield Things Devices Using Deep Eigenspace Learning," in *IEEE Transactions on Sustainable Computing* , 2019.

[23] H. HaddadPajouh, A. Dehghantanha and R. Khayami, "A deep Recurrent Neural Network based approach for Internet of Things malware threat hunting," *Future Generation Computer Systems*, 2018.

[24] J. Su, D. Vargas, S. Prasad, D. Sgandurra, Y. Feng and K. Sakurai, "Lightweight Classification of IoT Malware Based on Image Recognition," in *IEEE International Conference on Computer Software & Applications* , 2018.

[25] S. Homayoun, A. Dehghantanha, M. Ahmadzadeh, S. Hashemi and R. Khayami, "Know Abnormal, Find Evil: Frequent Pattern Mining for Ransomware Threat Hunting and Intelligence," in *IEEE Transactions of Emerging Topics in Computing* .

[26] 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*, 2009.

[27] "Virus Share," [Online]. Available: https://virusshare.com/. [Accessed 20 March 2020].

[28] "Package Search for Linux and Unix," [Online]. Available: https://pkgs.org/. [Accessed 20 March 2020].

[29] "IDA Pro," [Online]. Available: https://www.hex-rays.com/products/ida/. [Accessed 2 April 2020].

BIOGRAPHY

Aditi Atul Khare is a graduate student pursuing Master of Science in Computer Engineering at George Mason University, Fairfax, Virginia, USA. She graduated from Sir M. Visvesvaraya Institute of Technology, Bangalore, India with a Bachelor of Engineering degree in 2017. Her research interest lies in malware detection and classification.