# Effective Malware Detection using Shapely Boosting Algorithm

Rajesh Kumar, Geetha S
School of Computer Science and Engineering
Vellore Institute of Technology
Chennai, India

***Abstract*—Malware constitutes a prime exploitation tool to attack the vulnerabilities in software that lead to a threat to security. The number of malware gets generated as exploitation tools need effective methods to detect them. Machine learning methods are effective in detecting malware. The effectiveness of machine learning models can be increased by analyzing how the features that build the model contribute to the detection of malware. The model can be made robust by getting insight into how features contribute to each sample that is fed to a trained model. In this paper, the boosting machine learning model based on LightGBM is enhanced with Shapley value to detect the contribution of the top nine features for classification such as true positive or true negative and for misclassification such as false positive or false negative. This insight in the model can be used for effective and robust malware detection and to avoid wrong detections such as false positive and false negative. The comparison of the top features and their contribution in shapely value for each category of the sample gives insight and inductive learning into the model to know the reasons for misclassification. Inductive learning can be transformed into rules. The prediction by the trained model can be re-evaluated with such inductive learning and rules to ensure effective and robust prediction and avoid misclassification. The performance of models gives 98.48 at maximum and 97.45 at a minimum by 10 fold cross-validation.**

***Keywords—Artificial intelligence; machine learning; malware detection; shapely value; decision plot; waterfall plot***

## I. Introduction

At the current time, the malware is generated in large numbers. Open Threat Exchange [1] is a platform for the exchange of information related to computer security. The reason for the high volume generation of malware is both from the generation side, and end-use of it. Malware authors use tools such as polymorphic and metamorphic engines. Metamorphic engines can generate malware with minor modification of code. It uses techniques such as register reassignment, NOP instruction insertion, code transposition, the substitution of machine-level opcode/instructions, dead code insertion, and combinations of these techniques. Polymorphic engines can generate malware with encryption, prepend data, append data, and combinations of these techniques. The generated malware exhibits the same behavior as old malware. However, this generated malware can evade detection by antivirus software based on the signature. The detection engine of many antiviruses is based on the signature. Hence, databases of signatures need a constant update for upcoming malware. On the use side of malware, the number of software products has increased over time. Ten top software products with vulnerabilities are listed in Table I [2]. Software products with vulnerabilities from the top ten vendors are listed in Table II [3]. These vulnerabilities are exploited for an attack using existing or new malware. The software products are not limited to but include Operating Systems (OS), Driver for hardware devices, software applications, etc. The more a software product is used and popular, the more attacks it may have. Hence, hackers need more malware to attack the vulnerabilities. The vulnerabilities in hardware, OS, application, firewalls, anti-virus products, etc. may be by accident. The author [4] identifies three phases of the life cycle of vulnerabilities. In the first phase, a product is released in the market. The second phase starts when a vulnerability is found in the software product. In the third phase, the vulnerability has to be fixed by the developer and released for the user of the software. The vulnerabilities can de systematically discovered with needful tools. Knowing vulnerabilities is not enough, the vulnerabilities have to be proven by exploits, and attack software (malware). Machine learning and deep learning methods are used for malware detection and classification in research work these days.

TABLE I. TOP SOFTWARE VENDORS WITH VULNERABILITIES

| **SL. No.** | **Vendor Name** | **Number of Products** | **Number of Vulnerabilities** | **#Vulnerabilities/#Products** |
|---|---|---|---|---|
| 1 | Microsoft | 655 | 8178 | 12 |
| 2 | Oracle | 938 | 8043 | 9 |
| 3 | Google | 124 | 6571 | 53 |
| 4 | Debian | 106 | 5697 | 54 |
| 5 | Apple | 139 | 5380 | 39 |
| 6 | IBM | 1314 | 5334 | 4 |
| 7 | Cisco | 5592 | 4137 | 1 |
| 8 | Redhat | 407 | 3984 | 10 |
| 9 | Canonical | 49 | 3075 | 63 |
| 10 | Linux | 23 | 2751 | 120 |

TABLE II. TOP OPERATING SYSTEMS WITH VULNERABILITIES

| Sl. No. | Product Name | Vendor Name | Product Type | Number of Vulnerabilities |
|---|---|---|---|---|
| 1 | Debian Linux | Debian | OS | 5572 |
| 2 | Android | Google | OS | 3875 |
| 3 | Ubuntu Linux | Canonical | OS | 3036 |
| 4 | Mac Os X | Apple | OS | 2911 |
| 5 | Linux Kernel | Linux | OS | 2722 |
| 6 | Fedora | Fedoraproject | OS | 2538 |
| 7 | iPhone OS | Apple | OS | 2522 |
| 8 | Windows 10 | Microsoft | OS | 2459 |
| 9 | Windows Server 2016 | Microsoft | OS | 2233 |
| 10 | Windows 7 | Microsoft | OS | 1954 |

The objective of this paper is to further improve the effectiveness of the machine learning (ML) model based on boosting algorithms such as LightGBM by overcoming the wrong prediction, misclassification the ML model may have. Good ML models are made general with feature engineering and learning from a large dataset, to detect unknown malware. There are many algorithms for ML models and many feature engineering techniques to make the models effective that result~~ing~~ in increasing the accuracy of models. Misclassification in the machine learning model is wrong identification. For malware, the ML model may not identify them and they are termed as a false negative. A false negative detection can be very dangerous for any organization. As the malware is not detected, it will be able to meet the objective of the attacker despite all the security solutions applied. ML model may also declare benign software as malware. Such occurrences are termed false positives. A false positive detection causes issues such as panic among users of the software, inconveniences, non-use of software until a confirmed source declares the software as benign. All machine learning models have misclassification without exception.

Machine learning models to detect malware are many and they also use feature importance as part of an algorithm to identify top features. There are other methods for feature importance using feature engineering such as Principal Component Analysis (PCA), Redundant Feature Removal (RFR), and Haar Wavelet Transform (HWT) [6] and Leave One Feature Out importance (LOFO) method [7].

In this paper, a novel method is proposed to identify the change in top features that contribute to the misdetection of malware or future input sample that may be malware or benign software to a trained ML model. In addition, to identify the amount of contribution the top features are having for misclassification of a future sample in consideration as input to the ML model. Shapely values and visualization techniques are used to achieve these objectives. Shapely values are from classic game theory. Shapely values are used to find feature importance in an ML model. Lundberg et al. [5] have used Shapley value for explainable artificial intelligence. Hence, Shapley values can identify the top features in an ML model. The top features in an ML model based on LightGBM have shapely values associated with them. These top features along with their contribution to Shapley values are visualized using decision plot, waterfall plot, and force plots. Further, this work proposes to identify the false positive and false negative from the test dataset part. Further, the work also associates visualization with change in top features and amount of contribution of top features. Having identified top features and the amount of contribution of the top features for misclassification, this work proposes the use of inductive learning techniques to overcome the misclassification of future samples. The present work aims to improve the effectiveness of the ML model based on the LightGBM model. It can be used for zero-day malware detection as well.

The gaps that this work address~~es~~ are highlighted as follows.

- The~~se~~ feature importance from algorithms and feature engineering methods cannot associate the top features for a new sample used for prediction by a trained machine learning model.
- They cannot determine the amount of contribution of a feature for a sample used for prediction by a trained machine learning model. Hence, they cannot associate the visualizations with the amount of contribution of a feature for a new sample to be predicted by the machine learning model.
- There remains always a doubt if the new sample under test is part of high accuracy as published for the model or part of misclassification as false negative or false positive.
- The inductive method proposed in this work improves the probability of prediction to a higher level.
- A novel approach as proposed in this work is not available in the literature survey. Hence, this paper opens new dimensions for increasing the probability of effective detection of a new sample by a trained model.

Specific contributions in this study are:

- Use of Shapley values and visualization for identification of top features for false negative (FN), false positive (FP), true positive (TP), and true negative (TN) categories of samples for LightGBM machine learning models.
- Amount of contribution by top features for each predicted category in Shapley values are identified. So that the comparison for inductive learning is effective.
- Comparison of the features and amount of contributions of features for samples with the test dataset part that may be FP, FN, TP, and TN. Using the comparison to identify the top features and their contribution for misclassified FP and FN samples.
- Use of LightGBM, boosting algorithms, for effective prediction of a future sample that may be malware or benign software. The proposed inductive method will avoid misclassification and improve the effectiveness of the ML model.

This paper is organized with a literature survey in Section II, followed by the methodology of malware detection and the use of shapely values for visualization in Section III. The Dataset, experimental setup, and results are outlined in Section IV. The paper concludes in Section V with a conclusion and an Appendix in Section VI.

## II. Literature Survey

Malware is like any software product. It has to be distinguished from a benign software product. The methods available to distinguish and detect the malware are broadly categorized into static analysis, dynamic analysis, and hybrid analysis.

### A. *Static Analysis*

In static analysis, the malware is not executed. Features for machine learning are extracted from the software without running the software, the sample under consideration. It has the advantage that the sample cannot infect the system used for extraction of features. All the software, malware, shared libraries required, and dynamic link libraries (DLL) have a header. For windows, the header of the executable is termed Portable Executable (PE) header. The features from the PE header of windows executables can be extracted as explained in [6][8]. In addition, features can be extracted using properties of the executable file as an object and are termed file-related features. File related features are not limited to but include a histogram of bytes in executable, the entropy of complete file entropy of various parts of files, strings embedded in the executable, N-grams [9] from byte code, N-grams from assembly code, N-grams from API calls, images of hex bytecode of a file [10][11], images of hex bytecode of different part of a file, etc. Many machine learning models and deep learning models use features with different combinations derived from static analysis[12]. However, malware authors use methods such as obfuscation [13], encryption of various types to evade feature extraction methods. The obfuscation and encryption methods are many and may be categorized into standard and non-standard (private). These shortcomings of static analysis may be overcome by dynamic analysis [14].

Authors in [15] convert the sample file to images and extract features using the trained CNN model. The extracted features are plotted using t-Distributed Stochastic Neighbor to identify the cluster of malware. Subsequently, they make N-grams with n values 1 to 5 using the API call sequence for six types of malware actions. The malware actions are creating or modifying files, hooking on to system services, getting information for loading the DLL, etc. The N-grams are used with eight types of distance measurement to make a similarity matrix using four types of kernel functions with the Support Vector Method (SVM). Distance measurements used in this work are Cosine, Bray-Curtis, Canberra, Manhattan, Chebyshev, Euclidean, Hamming distance, and Correlation for feature extraction. This technique may handle malware with a known packing method, as they can be unpacked to process and get features but will have a deficiency in handling packed malware with unknown packing methods.

Yousefi-Azar et al in [15] extract static features of a sample, malware, or benign software, using term frequency based on natural language processing. Extracted features are used with the deep learning model and Extreme Learning Machine (ETM) for malware detection. Backpropagation results in large feature space which increases computation complexity. The authors multiply term frequency with a random projection matrix to reduce the computation complexity. Balanced android dataset Drebin and Dexshare and windows executables from 2016 are used as a dataset. Windows executables from 2017 are tested as zero-day malware to achieve an accuracy of 95.5%.

The authors [16] collect malware samples that are used for attacks in financial institutions in Brazil, affecting cyber users for over 6 years. They use static analysis to extract features from PE header of collected samples and use Multilayer Layer Perceptron, K-nearest neighbor (KNN), Random Forest (RF), and Support Vector Machine (SVM) classifiers to detect malware. Further, they identify the family of malware using the t-Distributed Stochastic Neighbor Embedding (SNE) method. Concept drift of ML model is detected using Drift Detection Method (DDM) and Early Drift Detection Method (EDDM) to detect drift in the malware samples over time. The authors visualize and relate the new malware families coming over time using confirmation and warning indicated by the drift methods. They conclude that a warning indication by drift methods implies a degradation of ML models and a confirmation indication by drift method implies that the ML model needs to be updated.

### B. *Dynamic Analysis*

In dynamic analysis, the malware is executed in a protected environment, and the behaviors, actions of malware are observed. In a normal environment, the sample will infect the system and will affect the future normal use of the system. Hence, a protected environment is used to avoid infection of the system conducting the malware test. The actions and behaviors of malware are not limited to but include adding, deleting, and modifying related changes in the file name, registry, processes, communication in the network, system configuration, etc. Features are derived with these changes and used in machine learning models with various algorithms. The dynamic analysis method is very expensive in terms of time to execute malware, computing resources, and trained manpower required. Besides, the malware authors employ techniques to avoid malware detection. One of the techniques employed by the malware author is to detect the virtual environment required for running the malware. If the virtual environment is detected, they switch off the behavior of malware and act as benign software. Another technique used by malware authors is to connect to the command and control center owned by them and download the malware at a later time to take control of the target machine. If the network is not available in virtual environment, the sample acts as benign software. Hence, trained persons are required to note this behavior of malware. The hybrid analysis is used to overcome these shortcomings of dynamic analysis.

Robert et al. [17] use a large dataset of malware with a Malheur tool to know the behavior of samples. Malheur tool executes the samples and generates a report. Needful information such as DLLs imported, API used as the callback are extracted from the report to understand the actions, behavior of malware with help of domain experts. Domain experts make rules and rules are externalized to the malware detection module. Authors believe malware will exhibit its behavior as per framed rule and that can be detected. However, new types of malware may not exhibit behavior as per rules framed, because that malware was not part of the dataset used. Hence, this unknown malware will not be detected.

Binayak et al. [18] create a knowledge database of In-memory processes based on the use of Dynamic Link Library (DLL) sequences using TF-IDF (Term Frequency-Inverse Document Frequency) and multinomial logistic regression based learning approach. The suspected process from malware uses a different DLL than of system DLL. This knowledge database is compared with DLL sequences used by In-memory processes to identify suspected, unwanted processes and malware.

### C. Hybrid Analysis

Hybrid analysis combines static and dynamic analysis to overcome their shortcoming. Lifan Xu et al. [19] extract both static and dynamic features from android malware dataset and represent the features as vector. Advance features are derived using deep learning, a Deep Neural Network (DNN) using both the original static and dynamic feature vector sets. The advanced and original features are concatenated as new vectors as input to the DNN that modifies with multiple different kernel to detect malware. The combined hybrid analysis has shortcomings as in dynamic analysis or static analysis.

Sethi et al. [20] use feature from both static analysis and dynamic analysis on PHP, pdf, exe files. For dynamic analysis, the authors use a Cuckoo sandbox. Cuckoo sandbox is a virtual environment to run executable. It gives an analysis report of actions and behavior of the file executed. J48, SMO, and Random Forest machine learning algorithms are applied in the WEKA tool with the combined feature extracted using static and dynamic analysis. They achieved 100% accuracy with J48.

The literature survey gives different methods of improving the accuracy and other performance parameters of the machine learning model by feature engineering for malware detection. However, they do not give insight into the top features and contribution of each feature for a new sample by a trained machine learning model. Hence, there is a gap in research that can give insight into the top features and their contribution in the prediction of an unseen sample by machine learning model. This work is an effort to fill the gap.

## III. Methodology

### A. Shapley Value and Feature Importance

The machine learning model should be both interpretable and accurate. Interpretation of ML model based on decision tree may be based on decision path, heuristic value to features, and model-agnostic. In this work, Shapley value is used for making the ML model interpretable. A local explanation is assigning a numeric measure, credit, to each input feature that constitutes a machine learning model based on a decision tree. These local explanations are combined to represent a global structure that represents an ML model based on a decision tree or an ensemble of decision trees. The ensemble of decision trees may be based on a bagging algorithm such as Random Forest or boosting algorithm such as LightGBM. The global explanation of the ML model continues to retain the local faithfulness as in local explanation. Shapely values from game theory satisfy simultaneously local accuracy, consistency, and missingness three properties required for credit score to a feature in an ML model. The credit score, Shapley values, are computed by one feature at a time into the output function of the model with some condition as in Eq. (1). Lundberg et al. [5] follow the causal do-notation formulation. It justifies use of the Shapley additive explanation (SHAP) interaction values as a richer type of local explanation and feature perturbation formulation.

$$f_x(S) = E[\, f(X) | do(X_s = x_s)] \tag{1}$$

S = Set of features to condition on

X = A random variable from M input features of model

x = input vector for the current prediction for the model

Lundberg et al. [5] give TreeExplainer, an explanation method for ML models based on tree, that enables optimal local explanations based on shapley values from classic game theory. Classic Shapley values are ways to measure feature importance. It is optimal and maintains natural properties from cooperative game theory. Exact computation of these values is NP-hard problem. Hence, they have approximate computation. Authors have developed an algorithm for decision tree categories of algorithms that computes local explanations with theoretical guarantees of local accuracy and consistency in polynomial time with Shapley values. Local explanations are also used to capture feature interactions in a theoretically grounded way. S is the set of features in Eq. (1) to condition on and refers to features of a specific tree in the ensemble of trees in the boosting LightGBM machine learning model. We can find the SHAP value for each feature, x in Eq. (1), in a tree using the TreeExplainer and add for all the features, X in Eq. (1), in the tree under consideration to match with the tree. This can be applied to all the trees in the model one by one. Finally, we find the contribution of a feature for the ensemble of trees by the TreeExplainer. By knowing the contribution of all features in an ML model, it provides valuable insight into top features for each prediction.

### B. Malware Detection Model

All samples in the dataset are from windows executable. The features are derived from the PE header of the samples and as properties of a file. Each window executable contains a PE header that is explained in [6] [8] [21]. The PE header can be extracted using a python program using "Library for Instrumenting Executable Files" (LIEF) a library in python. The extracted features are listed in detail in Appendix A. PE header consists of DOS header, file header, NT header, section header, optional header, and many directories such as Import directories, Resource directory, Export directory, and Exception directory. Import directories list Dynamic Link Libraries (DLL) loaded by the executable and Application Program Interfaces (APIs) used by executables. Resource directory lists the information required by executable such as icons, bitmaps, strings, menus, dialogs, configuration files, version information, etc. Exception directory lists exception handling information. Features extracted are listed in Appendix A. Some of the features are described here. File header of PE header gives features such as timestamp, vsize, has_debug, has_relocations, has_signature, has_tls, has_symbol, imports, Machine1-Machine10 listed in Appendix A. Machine representing, type of processor required, in the file header part of PE header is hashed and put into one of ten bins and named as Machine1 - Machine10. Features that are hashed and put in several bins are named like this. Section header and optional header give section name, section size, section characteristics, and start and end byte contents of each section. The section name is a string. It is hashed and put into 1 of 50 bins. This gives us feature entry_name1 - entry_name50 listed in Appendix A. Section size, section virtual size, and section characteristics values are hashed and put into 1 of 50 bins. These operations give us features Sec_size_1 – sec_size_50, sec_vsize1 - sec_vsize50, sec_char1 - sec_char50 listed in Appendix A. Entropy of content of each section in the sample is hashed and put in to 1 of 50 bins. This gives us features sec_entropy_1 - sec_entropy_50 listed in Appendix A.

DLL in an import directory and the name of an API in the DLL are concatenated to make a string. The string is hashed and put into one of 1280 bins. This gives us the feature Imp1-Imp1280 listed in Appendix A. Function name in the export directory is hashes and out into one of 128 bins. This gives us feature exp1-exp128 listed in Appendix A. File-related information used to derive features are histogram of bytes, strings, and entropy of hex values in each sample. The byte value in the sample can be 0-255. A histogram is count of value of the byte in each sample. The count of the value of a byte is put into the respective bin H1-H256 to represent the feature listed in Appendix A. Strings in a sample give very important, insightful information used by malware. Strings reveal created and modified filenames and registry related information. Strings may also reveal IP addresses used by malware authors for communication, command and control center URLs, signature of malware authors and groups. All strings of size five-character or more are extracted, hashed, and put in one of 104 bins. This gives us features Str1 - Str104 listed in Appendix A. The encryption and packing methods increase the entropy, disorder of bytes in samples. Entropy is computed as the method described by [8]. In this method, a block size of 2048 bytes is extracted and counts of bytes are put in 16x16 bins. These operations of making a block of 2048 bytes with windows of 1024 bytes and putting in 16x16 bins are repeated for the entire content of a sample. This gives us features Ben1-Ben256 listed in Appendix A. Both the PE header and file-related information give 2351 features. Dataset consists of malware and benign samples and belongs to January 2017 time period.

Gradient Boosting Decision Tree (GBDT) LightGBM ML algorithms are selected for experiments in this work. The ML algorithm is selected for the following advantages.

- Feature importance of the ML model can be extracted after training of the model.
- Faster training and prediction
- Ease of computation

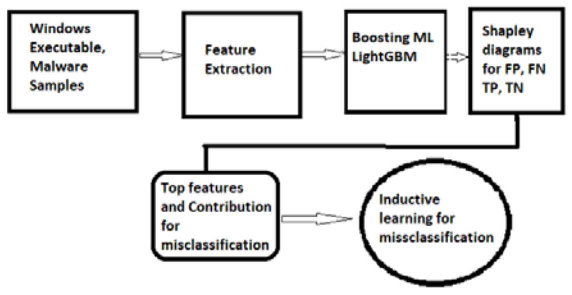
Windows Executable, Malware Samples
Feature Extraction
Boosting ML LightGBM
Shapley diagrams for FP, FN TP, TN
Top features and Contribution for misclassification
Inductive learning for missclassification

Fig. 1. System Block Diagram for Top Feature and their Contribution from Shapley Diagrams for Misclassification and Inductive Learning.

The system block diagram for this work is shown in Fig. 1. LightGBM boosting machine learning algorithm in the sklearn library is used to train the ML model. A trained model can predict the samples in the test dataset and correct detection of samples in true positive (TP), malware, and true negative (TN) benign software categories. Misclassified samples such as false positive (FP), benign software detected as malware, and false negative (FN), malware detected as benign software categories can also be identified. Nine to twenty five top features among the 2351 features can be identified for samples in TP, TN, FP, FN categories using diagrams such as waterfall plots, decision plots, and force plots. These diagrams show the amount of contribution by each top feature in Shapley values. Shapley values give a local explanation of top features with global structure as per ML model prediction for a sample. Changes in the top few features and their contribution to Shapley value for TP, TN, FP, and FN is compared. The comparison identifies a change in top features and their contribution for FP, FN also. This insight can be used as inductive learning to identify other samples which may have been misclassified and for future unknown samples without labels. Having found the misclassified samples by trained ML classifier, correct classification or malware detection can be performed. This leads to an increase in the performance of the trained ML model. One has to be very careful in this comparison and inductive learning with an unknown sample that does not have a label. Top features and their contribution in Shapley value for the unknown sample should match top features and their contribution in Shapley value for with known TP and TN samples also.

## IV. Experimental Results and Analysis

### A. Dataset

The dataset in the proposed system is derived from [21]. It has Malware data from December-2006 to December 2017. The dataset from December 2006 to December 2016 contains only the malware and no benign entries and the reason for exclusion. Fig. 2 shows the exclusions, filter and pre-process used on the dataset to get the sub dataset used in this experiment. Dataset part from January 2017 is used in this proposed system. The unidentified entries are without labels in the dataset and are excluded for malware detection and analysis. The unidentified entries in the dataset may be malware or cleanware. The dataset consists of 32761 malware and 17186 benign software that appeared in January-2017. The details of the derived dataset are in Table III.

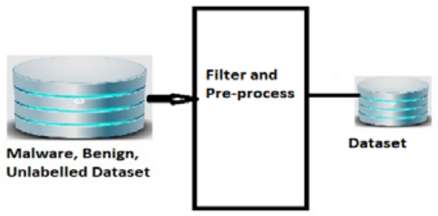

Fig. 2. Filter and Process the Database for the Experiments.

TABLE III. DATASET USED IN THIS WORK

| SL No. | Samples | label | appeared |
|---|---|---|---|
| 1 | 28606 | Unidentified | 2017-01 |
| 2 | 17180 | Benign | 2017-01 |
| 3 | 32761 | Malware | 2017-01 |

Each entry in the dataset has 2351 features. These features are from PE headers, sections of windows executable, systems APIs used in the executable, exported API from the executable, and file related properties. File related properties include Histogram of the complete executable in 256 bins, Byte entropy of executable file hashed into 256 bins and strings in 104 bins. The executable here means both the malware and cleanware. These features are defined in Appendix A and are used in the various diagrams in this paper. These features' names help identify exact features that are contributing to the detection of malware and the amount of contribution in the detection of malware or cleanware.

## B. Experimental Setup

Intel(R) Core(TM) i5-7200U CPU @ 2.50 GHz, 2701 MHz, 2 Core(s), and 4 Logical Processor(s) with 8 GB Ram is used as computing resource in this work.

## C. Malware Detection with LightGBM

Dataset is divided into a training set and testing set in the ratio of 70% and 30%. The model is trained with the training set and tested with the testing set. The results of this are in row 1 of Table IV. It has performance data for Accuracy, Precision, Recall, F1-score, and confusion matrix parameters in terms of false negative (FN), false positive (FP), true positive (TP), and true negative (TN).

30% of the dataset is separated for the testing of the LightGBM model. The samples in the test dataset are identified in false negative (FN), false positive (FP), true positive (TP), and true negative (TN) categories. It is interesting to explore how the top features and other features contribute to FP, FN, TP, and TN samples by the LightGBM algorithm. Waterfall plot, decision plot, and force plot are drawn with Shapley values.

TABLE IV. PERFORMANCE OF LIGHTGBM MODEL WITH ZERO-DAY MALWARE

| | Accuracy | TP | FP | FN | TN | precision | recall | f1-score | support |
|---|---|---|---|---|---|---|---|---|---|
| D1-Test | 98.483 | 5520 | 150 | 100 | 10711 | 0.99 | 0.99 | 0.99 | 10811 |
| D1 | 99.389 | 16998 | 182 | 123 | 32638 | 0.99 | 1 | 1 | 32761 |

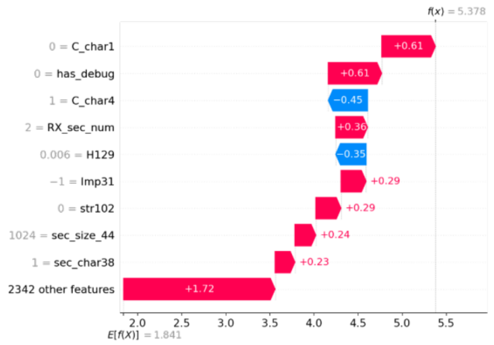

Fig. 3. Waterfall Plot of True Positive Sample in a Dataset with Shapley Values for each Feature.

Fig. 3 shows the waterfall plot for a true positive in the dataset. The Shapley value sum features to 5.378. The waterfall plot adds the contribution of each feature and also shows the top features with their contribution leading to the decision. Although the total contribution of 2342 features; lowest bar, in figure is significant compared to any top feature. In additional analysis, the feature importance of LightGBM showed only 588 features contributed to the model. Other 1763 features do not have any contribution to malware detection. Hence, many features in 2342 features have zero contribution. Kumar et al. [22] identified that 276 features among 2351 only contributed to prediction in the XGBoost model with 600k samples of training dataset from [21]. The remaining 2075 features have zero contribution to the model.

These figures help us to identify top features contributing to decision at leaf note with LightGBM algorithm.

Fig. 4 is displays the decision plot for a true positive entry in the dataset and shows how the top features contribute to make the decision. The decision plot adds the contribution of each feature and draws the line that takes it to make a decision. It has the same TOP features as in Fig. 3. The waterfall plot adds the contribution of all the features but does not show the graph that leads to a decision as in the decision plot.

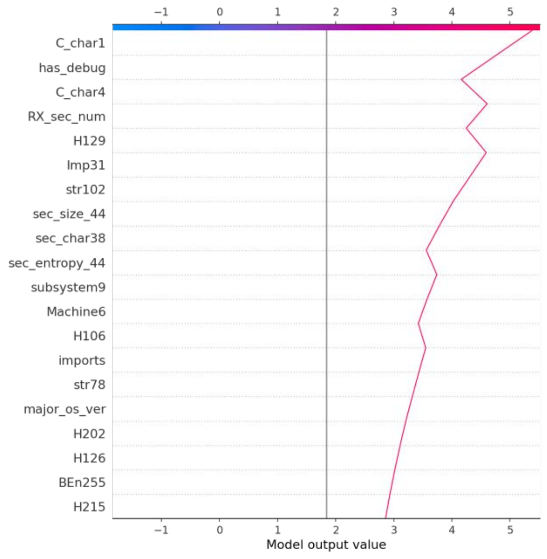

Fig. 4. Decision Plot of True Positive Sample in Dataset with Shapley Values for each Feature.

The decision plot can be for more number of samples. Fig. 5 shows the decision plot for the first ten samples in the dataset. The value that shows negative from zero in blue color are benign and the values that are on the right side of two, the purple color vertical line, in the graph are malware. The seven samples, pink color, with the specific features as shown are the malware and three samples are benign. The objective of these figures is to display how the features are contributing to decision with use of the LightGBM model. The label of samples is verified with the prediction of each sample with the LightGBM GBDT algorithm for all the 10 samples. It matches as given in the decision plot.

Fig. 6 shows the force plot for a true positive sample in the dataset in the Shapley values. The force plot shows how each of the feature is contributing in the positive direction from the left side with red color and other features that contribute negatively to scale value down and finally the value set near 5.38 for Shapley values. The top three features that contribute to the decision are named. The meaning of these features can be referred at Appendix A. The force plot cannot display the name of many features as in the decision plot. The top three features are the same as in Fig. 3 and Fig. 4.

Fig. 7 shows the force plot for the first 10 samples of the dataset in Shapley values. This figure is like rotating Fig. 6 clockwise and stacking the ten force plot of the figure in the x-axis. The count of the sample is seen at the top with numbers 0, 1, 9. The Y-axis displays the Shapley value for samples. The Shapley values for sample 2 are different from another sample. This change in Shapley value for each sample in the dataset is visible.  It is possible to change the parameters in x-axis and y-axis from a drop down menu and analyze the top features for a sample. Amount of contribution top features make to the decision using the LightGBM algorithm can be observed.

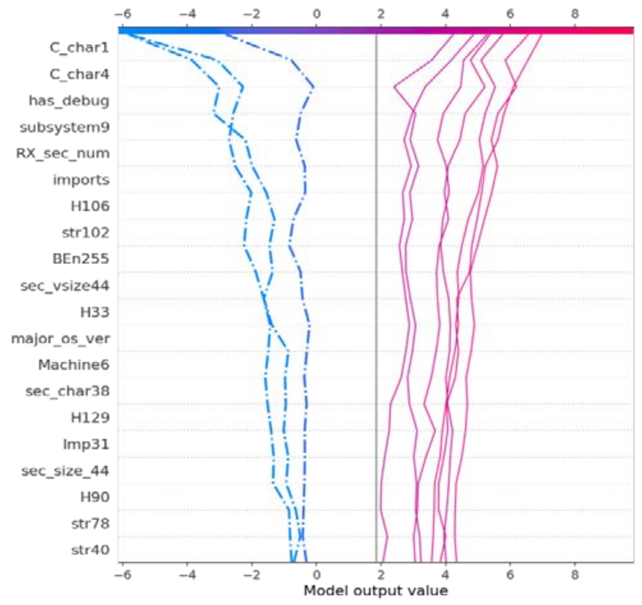

Fig. 5. Decision Plot of First 10 Samples in Dataset with Shapley Values for each Feature.

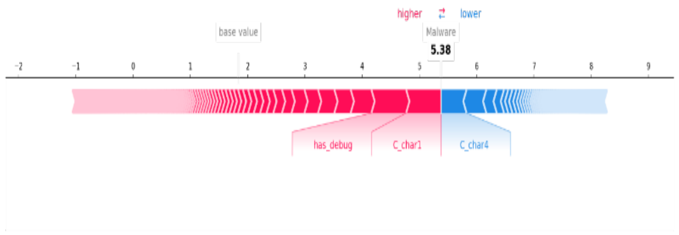

Fig. 6. Force Plot of True Positive in Dataset with Shapley Values for each Feature.

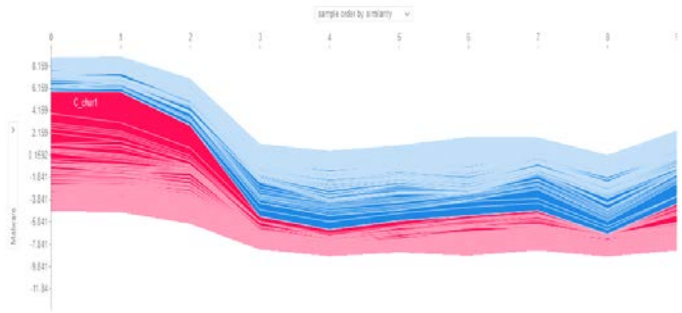

Fig. 7. Force Plot of First 10 Samples in Dataset with Shapley Value for each Feature.

Fig. 8 displays a waterfall plot for a false positive sample from the test dataset part of the dataset. The advantages of waterfall plots are:

- Top 9 features contributing to predicting the sample as a false positive.
- In the Shapley scale, it starts at 2.0 and adds up to 3.589 with the top 9 features contributing in both positive and negative directions.
- The contribution of the remaining 2342 features for the sample is +0.2, much less than the top five features.

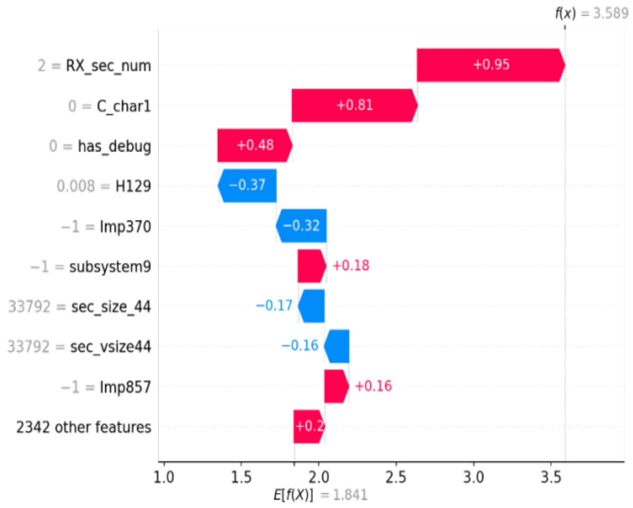

Fig. 8. Waterfall Plot of a False Positive Sample in Dataset in Shapley Value for each Feature.

Fig. 9 reveals the force plot for a false positive sample in the test dataset from dataset. The top features are RX_sec_num, C_char1, and has_debug from the PE header. The contribution of RX_sec_num is the highest among all the features. The top features for false positive in Fig. 9 are very different from the top features of malware (true positive) in Fig. 3. In addition, the final Shapley value for the sample is down to -0.09 in Fig. 9 compared to 5.38 in Fig. 3 for malware. The start point is very low at less than -6 in Fig. 9 compared to the start point at -1 in Fig. 3.

Fig. 10 presents a waterfall plot for a False Negative sample in the test dataset from the dataset in the Shapley value. All the advantages as explained for a false positive sample can be observed. In addition, features that are making the sample false positive and false negative can be compared. There is no contribution of the RX_sec_num, H129 feature in the false negative sample as in the false positive sample. The contribution of feature C_char4 is there in false negative but not in false positive.

Fig. 11 shows the force plot for a False Negative (FN) sample in the test dataset in Shapley values.

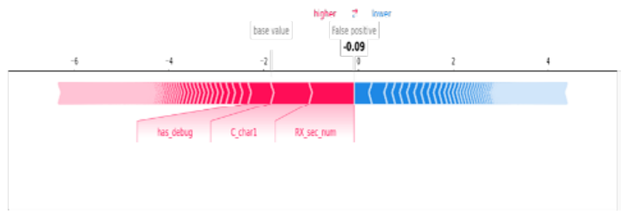

Fig. 9. Force Plot of a False Positive Sample in Dataset in Shapley Value for each Feature.

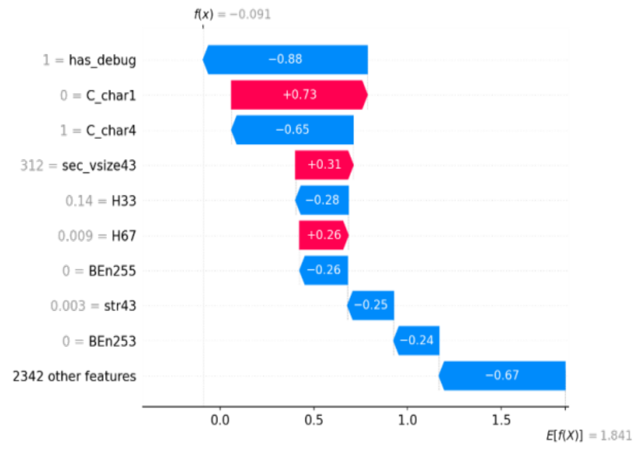

Fig. 10. Waterfall Plot of a False Negative Sample in Dataset in Shapley Value for each Feature.

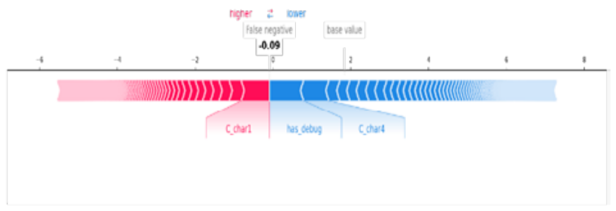

Fig. 11. Force Plot of a False Negative Sample in Dataset in Shapley Value for each Feature.

Fig. 12 shows the waterfall plot for the True negative sample in test dataset in Shapley value. Fig. 13 gives the force plot for the True negative (TN) sample in the test dataset in Shapley value.

The top features of waterfall plots in Shapley value from Fig. 3, Fig. 8, Fig. 10, Fig. 12 for in FP, FN, TP, and TN samples respectively in test dataset of the dataset are compared in Table V. Top features are listed in the features column. For a sample in each category in false positive, false negative, true positive and true negative, it identifies the presence of a feature as “Y” and no presence as “N”. Further, it identifies the topmost feature, with the value among the top feature with a “T” in each category. The probability value contributed by each feature is identified in respective columns. This table helps to conclude that there is disjoint set of features for each category samples in FP, FN, TP, and TN. The topmost feature for the FN sample is has_debug and is present in FP and TP. The topmost feature for FP is Rx_sec_num and contributes very low value in other categories of samples.

The contribution of the remaining 2342 features is lowered significantly for FP and FN. For TP the value is +ve .42, for TN the value is negative -.03.

These comparisons can identify the misclassified FP and FN samples and improve the efficiency of the ML model by correct classification for an unknown sample. Few insightful rules that can be formed are as follows:

- The malware sample with a high contribution of Imp321, H33, C_char1, and str43 may be a FP sample.
- The Malware sample with the highest contribution by Rx_sec_num among all the features will be a FN sample.

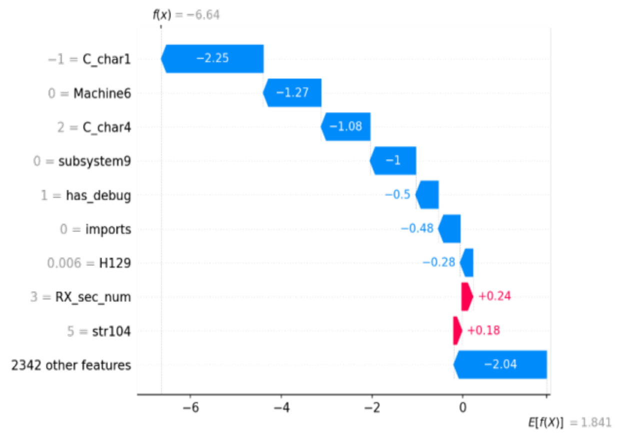

Fig. 12. Waterfall Plot of a True Negative Sample in dataset in Shapley Value for each Feature.

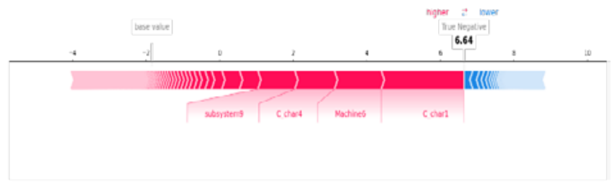

Fig. 13. Force Plot of a True Negative Sample in the Dataset in Shapley Value for each Feature.

TABLE V. FEATURES AND THEIR CONTRIBUTION IN FALSE POSITIVE, FALSE NEGATIVE, TRUE POSITIVE, TRUE NEGATIVE PREDICTION BY LIGHTGBM MODEL

| S L no | Features | False Positive | False Negative | True Positive | True Negative |
|---|---|---|---|---|---|
| 1 | C_char1 | Y,+0.81 | Y, 0.73 | Y,**0.61** | Y, **–2.25** |
| 2 | C_char4 | N | Y, -0.65 | Y, -0.45 | Y, - 1.08 |
| 3 | Machine6 | N | N | N | Y, -1.27 |
| 4 | Subsystem9 | 0.18 | N | N | Y, -1.0 |
| 5 | Rx_sec_num | Y, **+0.95** | N | Y, 0.36 | 0.24 |
| 6 | has_debug | Y, +0.48 | Y, **-0.88** | Y, 0.61 | -0.5 |
| 7 | sec_size43 | Sec_size44 = -0.17 | Y, 0.31 | 0.24, sec_char38 | N |
| 8 | H129 | Y, -0.37 | Y, H33 = -0.28 H67=0.26 | Y, -0.35 | -0.28 |
| 9 | Imp370 | Y, -0.32 imp857=0.16 | N | Imp31=0.29 | N |
| 10 | sec_char38 | N | N | 0.23 | N |
| 11 | Str43 | N | -0.25 | N | str104=0.18 |
| 12 | imports | N | N | N | -0.48 |
| 13 | Ben255 | N | -0.26, BEn253= -0.24 | N | N |
| 14 | Other 2342 features | 0.2 | Y,-0.67 | Y, 1.72 | Y, -2.04 |

## D. *k-Fold Cross-Validation*

Cross-validation with k=10 is performed for less biased and less optimistic accuracy value for the LightGBM model. The test dataset is used for this 10-fold cross-validation test. The results of cross-validation are tabulated in Table VI.

TABLE VI. TEN K FOLD CROSS-VALIDATION FOR THE TEST DATASET

| Sl. no | Accuracy |
|---|---|
| 1 | 0.97938144 |
| 2 | 0.9836165 |
| 3 | 0.97936893 |
| 4 | 0.9848301 |
| 5 | 0.97451456 |
| 6 | 0.97815534 |
| 7 | 0.97512136 |
| 8 | 0.97936893 |
| 9 | 0.9836165 |
| 10 | 0.97815534 |

## E. *Comparison with other Malware Detection Works*

This work is compared with other malware detection works in Table VII. This work achieves higher accuracy with datasets compared toYousefi-Azar et al. and comparable accuracy with Venkatraman et al. and Alazab et al. Jung et al [24] take 333 malware files with .swf extension into the test dataset from 2007-2015 for zero-day malware to get 51–100 % accuracy. Alazab et al. [25] get marginal higher accuracy of 98.6 compared to 98.49%. Shafiq et al. have a small size dataset and give the model performance at 99.2 Area under curve (AUC) that cannot be compared with accuracy.

They use more than three times malware compared to benign software for training and testing. They do not define ways to determine unknown malware. [6] Use only one tenth of benign software compared to malware. This highly unbalanced dataset lowers the probability of false positives. They consider zero-day malware as one which does not match known signature or unknown malware.

TABLE VII. COMPARISON WITH OTHER ZERO-DAY MALWARE WORKS

| Paper | Method | Sample/Dataset | Result/ Accuracy |
|---|---|---|---|
| This work | Boosting algorithms: LightGBM | Dataset Details in Table III | 98.49 |
| Yousefi-Azar et al. [15] | NLP and the term frequency tf-simhasing: term frequency of sample multiple with rand projection matrix | Android:Drebin, DexShare Windows PE files: Training:11983 Malware, 8912 Benign (2016) Testing: 12127 Malware, 11983 | 97.33 |
| Venkatraman et al. [23] | Malware files to image as input to pre-trained CNN to get features, Apply SVM with SMO-Normalized Polynomial | 52k samples | 98.6% |
| Jung et al. [24] | API call sequence features Use Deep Feed-forward NN, RNN | Malicious .swf files 333 Benign .swf files 333 | 51% to 100% |
| Alazab et al. [25] | NB, kNN, 4 kernels with SMO. SMO–PolyKernel, SMO –Puk, SMO-Normalized, and SMO- RBF Backpropagation J48 and Neural Networks Algorithm | 66703 samples with 51223 Malware and 15480 Benign software | 98.6 |
| Shafiq et al. [6] | Ripper, Ibk and SVM-SMO classifier | 1447 Benign software 8892 + 5586 = 14478 malware | 99.2% g AUC |

## V. Conclusion

In this work, a boosting machine model based on LightGBM is enhanced using Shapely value to build an effective and robust machine learning model. Features derived by static analysis of malware and benign samples in the dataset are used to build the LightGBM boosting machine learning model. Datasets from Jan 2017 for malware is used for training and prediction. Waterfall plots, Decision plots and Force plots based on Shapley value helped identify the top few features. The Waterfall plots demonstrated a change in features and their contribution for a sample from different categories of samples as insight into the ML model. Table V compared the top features contributed to misclassified samples. The top feature for samples that is detected as false positive, false negative samples by trained models is analyzed and inductive learning rules are made. The inductive learning rules can be applied to unknown, unlabeled samples to avoid misclassification into FP and FN and to ensure correct detection. These top features and their contribution may be used to overcome the misclassification of malware. The cross-validation with the test dataset is 98.48 at maximum and 97.45 at minimum.

The work can be further extended to analyze change in features and to derive inductive learning rules for misclassification by other ML models for false positive and false negative cases to ensure correct prediction. The Shapley values for a feature may be mapped to the probability score of the ML model. This will help to correlate the Shapley value to probability value for a feature as local explanation and as a whole for a sample at global explanation (structure). Large datasets may be used to make a robust ML model and analyze reasons for misclassification for various families of malware such as ransomware, rootkit, Trojan horse, etc.

## References

[1] "AlienVault - Open Threat Exchange." https://otx.alienvault.com/ (accessed Dec. 29, 2021).

[2] "Top 50 products having highest number of cve security vulnerabilities." https://www.cvedetails.com/top-50-products.php (accessed Dec. 29, 2021).

[3] "Top 50 Vendors By Total Number Of 'Distinct' Vulnerabilities." https://www.cvedetails.com/top-50-vendors.php (accessed Dec. 29, 2021).

[4] H. Pohl, "Zero-Day and Less-Than-Zero-Day Vulnerabilities and Exploits," 2008.

[5] S. M. Lundberg et al., "From local explanations to global understanding with explainable AI for trees," Nature Machine Intelligence, vol. 2, no. 1, pp. 56–67, 2020, doi: 10.1038/s42256-019-0138-9.

[6] M. Z. Shafiq, S. M. Tabish, F. Mirza, and M. Farooq, "A Framework for Efficient Mining of Structural Information to Detect Zero-Day Malicious Portable Executables," no. October, 2015.

[7] S. A. Roseline, S. Geetha, and S. Member, "High Performance Android Malware Detection System using Gradient Boosting based Static Feature Selection and Classifier Paradigm," pp. 1–25.

[8] Stamp, Mark, Mamoun Alazab, and Andrii Shalaginov. Malware Analysis Using Artificial Intelligence and Deep Learning. Springer, 2021.

[9] J. Saxe and K. Berlin, "Deep neural network based malware detection using two dimensional binary program features," 2015 10th International Conference on Malicious and Unwanted Software, MALWARE 2015, pp. 11–20, 2016, doi: 10.1109/MALWARE.2015.7413680.

[10] S. A. Roseline, S. Geetha, S. Kadry, and Y. Nam, "Intelligent Vision-Based Malware Detection and Classification Using Deep Random Forest Paradigm," IEEE Access, vol. 8, pp. 206303–206324, 2020, doi: 10.1109/ACCESS.2020.3036491.

[11] J. Hemalatha, S. A. Roseline, S. Geetha, S. Kadry, and R. Damaševičius, "An efficient densenet ‑ based deep learning model for Malware detection," MDPI Entropy, vol. 23, no. 3, pp. 1–23, 2021, doi: 10.3390/e23030344.

[12] S. K. J. Rizvi, W. Aslam, M. Shahzad, S. Saleem, and M. M. Fraz, "PROUD-MAL: static analysis-based progressive framework for deep unsupervised malware classification of windows portable executable," Complex & Intelligent Systems, Oct. 2021, doi: 10.1007/s40747-021-00560-1.

[13] D. Mafaz, "Generic Packing Detection Using Several Complexity Analysis for Accurate Malware Detection," International Journal of Advanced Computer Science and Applications, vol. 5, no. 1, pp. 7–14, 2014, doi: 10.14569/ijacsa.2014.050102.

[14] M. Tang and Q. Qian, "Dynamic API call sequence visualisation for malware classification," IET Information Security, vol. 13, no. 4, pp. 367–377, 2019, doi: 10.1049/iet-ifs.2018.5268.

[15] M. Yousefi-Azar, L. G. C. Hamey, V. Varadharajan, and S. Chen, "Malytics: A malware detection scheme," IEEE Access, vol. 6, pp. 49418–49431, 2018, doi: 10.1109/ACCESS.2018.2864871.

[16] F. Ceschin, F. Pinage, M. Castilho, D. Menotti, L. S. Oliveira, and A. Gregio, "The Need for Speed: An Analysis of Brazilian Malware Classifers," IEEE Security and Privacy, vol. 16, no. 6, pp. 31–41, 2019, doi: 10.1109/MSEC.2018.2875369.

[17] R. Gove, J. Saxe, S. Gold, A. Long, G. B. I. Labs, and Z. Piper, "SEEM: A scalable visualization for comparing multiple large sets of attributes for malware analysis," ACM International Conference Proceeding Series, vol. 10-Novembe, pp. 72–79, 2014, doi: 10.1145/2671491.2671496.

[18] B. Panda and S. N. Tripathy, "Detection of Anomalous In-Memory Process based on DLL Sequence," International Journal of Advanced Computer Science and Applications, vol. 11, no. 10, pp. 185–194, 2020, doi: 10.14569/IJACSA.2020.0111025.

[19] L. Xu, D. Zhang, N. Jayasena, and J. Cavazos, "HADM: Hybrid Analysis for Detection of Malware," Lecture Notes in Networks and Systems, vol. 16, pp. 702–724, 2018, doi: 10.1007/978-3-319-56991-8_51.

[20] K. Sethi, B. K. Tripathy, S. K. Chaudhary, and P. Bera, "A Novel Malware Analysis for Malware Detection and Classification using Machine Learning Algorithms," ACM International Conference Proceeding Series, pp. 107–116, Oct. 2017, doi: 10.1145/3136825.3136883.

[21] H. S. Anderson and P. Roth, "EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models," 2018.

[22] Kumar, Rajesh; Geetha S, "Malware classification using XGboost-Gradient Boosted Decision Tree," Advances in Science, Technology and Engineering Systems Journal, Sep. 2020, doi: 10.25046/aj050566.

[23] S. Venkatraman and M. Alazab, "Use of Data Visualisation for Zero-Day Malware Detection," Security and Communication Networks, vol. 2018, 2018, doi: 10.1155/2018/1728303.

[24] W. Jung and S. Kim, "Poster : Deep Learning for Zero-day Flash Malware Detection," In Proceedings of the IEEE Symposium on Security and Privacy (S&P), pp. 2–3, 2015.

[25] M. Alazab, S. Venkatraman, P. Watters, and M. Alazab, "Zero-day malware detection based on supervised learning algorithms of API call signatures," Conferences in Research and Practice in Information Technology Series, vol. 121, no. June 2014, pp. 171–182, 2010.

## Appendix A

Bn1-Bn256: The entropy of executable for a window size of 2048 bytes is computed for the joint distribution of byte value and put into 16x16 bins. This is repeated for a step size of 1024 for the full file.

C_char1 - C_char10: Characteristics of the sample from PE file header to indicate if the file is DLL, Executable, systems file, etc. The value is hashed and put into one of the ten bins.

dll_c1 – dll_c10: The DLL characteristics value from the optional header of PE for the sample is hashed put into one of the ten bins.

entry_name1 - entry_name50: The name of each section in the PE header of the sample is hashed and put into one of the fifty bins.

exp1 – esp128: The exported APIs in the sample are hashed and put into one of the 128 bins.

Exports: Flag interpreted by LIEF, a python package, indicating the executable exports API in the data directory of PE header.

Imp1 – Imp1280: The DLL names and imported APIs in the DLL are hashed and put into one of the 1280 bins.

H1-H256: Byte count of hex value 0x00 to 0xFF of benign software, malware is put their respective bin. These counts are further normalized with the file size.\

has_debug: A flag in the characteristics field of file header in PE header of the sample, indicating debug information for the sample.

has_relocations: A flag in the characteristics field to indicate relocation sections, relocation directory.

has_resources: A flag in the characteristics field to indicate a resource section, a resource data directory.

has_signature: A flag in the characteristics field to indicate digital signature related information.

has_tls: A flag in the characteristics field to indicate tls section, tls data directory.

has_symbol: A flag in the characteristics field to indicate debug section with symbols.

Imports: Flag interpreted by LIEF, a python package, indicating the executable has imports of API from DLL.

Machine1-Machine10: It indicates hardware architecture 32/64 bit, processor for executable. The values are hashed and put in to one of the ten bins.

Magic1 – Magic10: Magic value from optional header of PE for the sample is hashed and put in to one of the ten bins.

num_of_sec_morethan0: Number of sections in section part of PE header which has content and size greater than 0 size.

Num_sec_noname: Number of sections in section part of PE header which are without a name. Generally, the name of a section is .text, .rdata, .data etc.

RX_sec_num: Number of sections in section part of PE header which has read and execute permission.

Sec_size_1 – sec_size_50: The size of each section in the PE header of the sample is hash and put in to one of the fifty bins.

sec_entropy_1 -- sec_entropy_50: The entropy of each section in the PE header of the sample is computed, hashed, and put into one of the fifty bins.

sec_vsize1 -- sec_vsize50: The memory size of each section in the PE header of the sample is hashed and put in to one of the fifty bins.

sec_char1 - sec_char50: The characteristics of each section in the PE header of the sample is hashed and put in to one of the fifty bins.

Size: Size of executable.

Str1-Str104: Five or more printable characters in the samples are hashed in to 104 bins. These strings include URLs starting with HTTP: HTTPS: registry keys starting with HKEYS, paths in systems such as c: /, file name, malware author’s messages, etc.

Subsystem1 – Subsystem10: Subsystem value from optional header of PE header of the sample. The values are hashed and put in to one of the ten bins.

timestamp: Date, the timestamp of a sample.

Vsize: virtual size of executable in memory.

W_sec_num: Number of section in section part of PE header which has write permission.