Sareen Fathima, et. al. International Journal of Engineering Research and Applications
www.ijera.com
ISSN: 2248-9622, Vol. 10, Issue 12, (Series-III) December 2020, pp. 64-68

RESEARCH ARTICLE OPEN ACCESS

Comparative Analysis of Malware Classification using Machine Learning Algorithms

[1]Sareen Fathima, [2]Suzaifa, [3]Abdul Khader
[123]Department of CSE, BIT Mangalore, India

Abstract - The rapid development of computer networks in the past decades has created many security problems related to malware on network systems. Malware can create successful attempts to cause the damage to the computer networks by unauthorized malwares. From the previous research we have found that it is easy to classify the known malware attack, but quite difficult to detect new malware and modify anomalous malware attack. The huge amount of data is due primarily to malware author's polymorphism. In order to effectively examine such data, all malware data belonging to the same family (class) should be found. In our case, it analyses the behaviour of data, and then this data is considered as normal based on the built model behaviour. Most of the existing malware classification systems rely heavily on human analysts to measure Log loss to differentiate between malware. With the increase in network traffic, manual work by humans in the classification system is a non-trivial problem. Thus, machine learning techniques are fast emerging, where we can train the system and even detect anomaly attacks. Microsoft Kaggle dataset has been used to train the model. Random Forest, k-Nearest Neighbor, Random Forest, Logistic Regression, and XGBoost Classifier are used.

Keywords— Machine Learning, Malware, Malware Classification, Microsoft Kaggle Dataset, XGBoost Classifier, Random Forest, k-Nearest Neighbor, Logistic Regression, LogLoss

---

Date of Submission: 14-12-2020 Date of Acceptance: 28-12-2020

I. INTRODUCTION

Malware is defined as any action that tries to gain unauthorized access to systems, do data manipulation, or render the system unstable by exploiting the existing vulnerabilities in the system. Malware volume is enormous and fast growing. The high volume of various unidentified files is one of the reasons why the malware authors introduce polymorphism. Due to these obscure tactics, malware files of the same class may look different. We need to be able to identify these data in the class to be successful in examining such malware data. The classification can be applied to new files found to be malicious and of some family. Because of the scope of the function, anti-virus organizations use computer preparation to identify and recognize malware. The objective of this project is to classify the malware into nine different classes of malware. The system predicts the likelihood of belonging to each class (soft labeling) instead of making a hard labeling. Malware can be analyzed using two methods.

1. Static Code Analysis.
2. Dynamic Behavior Analysis.

Static Code Analysis, Static code analysis involves the study of the binary file and the search for patterns in its structure indicating malicious behaviors. The fact that the malware writers can override detection methods through techniques such as metamorphic and polymorphic code obfuscation (dead code instruction) and polymorphism have been less effective in recent years because of the fact. In addition, Packers disrupt the entire program and make it necessary to evaluate the application only during execution time.

Dynamic Behavioural Analysis (DBA), involve executing a binary in an emulated or virtual machine environment and looking for patterns for requesting operating system (OS) or general system behavior that show malicious behaviour. Behavioral analysis has become more popular since it actually runs malware in its favorite environment, which makes it more difficult to completely avoid detection. The analysis evaluated will be performed such that details on the actions of the subject may be collected when driving in order to administer behavioral research. This data may be used to train an automatic distinction to distinguish harmful and benevolent applications. This data is then used. One of the most popular literature tools to understand malware behavior is to capture OS calls i.e. system calls.

In applications across many platforms including smartphones and devices, machine learning techniques and data mining have provided promising results for detecting the hidden malware effectively.

A number of static malware detection approaches have differentiated their work by studying different classifiers such as k-Nearest Neighbor (KNN), support vector machine (SVM), and Naıve Bayes (NB) etc. The findings indicate that XGBoost classification can achieve greater than 90% precision. Recent studies often use different data mining methods to evaluate authorization use for smartphone devices. In this study, we have taken machine learning and similitude mining approaches that focus on visualizing static and dynamic malware detection.

Some of the recent studies have studied visualization techniques to significantly accelerate the malware detection process. Visual analysis adapts to big data environments where data analysis includes complex data to integrate computation and human professional analytical reasoning. Similarity mining is a machine learning technique based on the analysis of similarities in distance measurement in visual analytics and has recently been adopted for malware detection. In this project, we demonstrate the similarity matrix between different malware programs widely used by attackers.

II. LITERATURE SURVEY

There are many works done in the area of malware classification.

K. Allix, Q. Jerome, T.F. Bissyande, J. Klein, R. State and Y.L. Traon (2014), the writers conducted a large collection of malware analysis and safe Android platform applications in this article. While a wide range of research has covered Android malware in recent years, none have addressed it forensically. Authors have analyzed a number of malware applications for deep study

P.V. Shijo and A. Salim (2015), this paper provides for the analysis and classification of an unknown executable file with integrated static and dynamic analysis methods. The method uses machine learning to train well-known malware and benign programs. By analyzing both binary code and dynamic behaviour, the feature vector is selected. The approach suggested incorporates the advantages of static as well as dynamic analysis, increasing performance and the outcome of classification.

Buczak AL and Guven E, (2016), this paper describes a focused literature survey of cyber analytics methods used in Machine Learning (ML) and data mining (DM) to support malware detection. Since data are so important in ML / DM approaches, several well-known cyber data sets used in ml / dm have been described. The complexity of the ML / DM algorithms is addressed, the challenges for cyber security using ML / DM are discussed and some recommendations are made as to when using a given method.

NayanZalavadiya, et. al., (2017), this paper examines the types of malware, tools, techniques and analysis of specific malware that is Trojan Remote Access (RAT). It gives full system access and monitoring to the remote system for malicious activity. RAT is very harmful malware. The authors say techniques of dynamic detection are more effective and the best way to analyze dynamic malware is to sandbox the environment.

Shalaginov A, Banin S, Dehghantanha A and Franke K. (2018), the rapid increase in the range and number of malware species made it very hard for forensic researchers to respond quickly. Machine Learning (ML) has therefore become a necessity to automate various aspects of the investigation of statically and dynamically controlled malware. We believe that static analysis supported by machine learning can be employed rather than resource-consuming, dynamic malware analysis, as a methodological approach in the technical Cyber Threats Intelligence (CTI) system.

Omar Al-Jarrah in (2019), proposed paper which uses artificial neural network to recognize the temporal behavior of malware attacks. The outputs are used by the pattern recognition neural networks to recognize the attacks, which are classified, by the classifier to generate attack alerts.

Peng Wei, Yufeng Li, Zhen Zhang et al. in (2020), describes Optimization method The C3.0algorithm. This algorithm shortens the average detection time by at least 24.69% on the premise of increasing the average training time by 6.9%; compared with the tested malware classification algorithms.

Royi Ronen, Marian Radu, Corina Feuerstein, Elad Yom-Tov, and Mansour Ahmadi (2018), provide a high-level comparison of the publications citing the dataset (Microsoft Kaggle Challenge). The comparison simplifies finding potential research directions in this field and future performance evaluation of the dataset.

III. OBJECTIVES OF MALWARE CLASSIFICATION

The objectives are:

1. To investigate on how to implement machine learning techniques to malware classification, in order to classify unknown malware.
2. To develop malware classification software that implements machine learning to detect unknown malware using Random Forest, KNN, Logistic Regression, and XGBoost algorithms.
3. To investigate the machine learning technique for malware classification using Microsoft Kaggle Dataset, and to achievs a high accuracy rate by obtaining the least log loss.

IV. PROPOSED SYSTEM

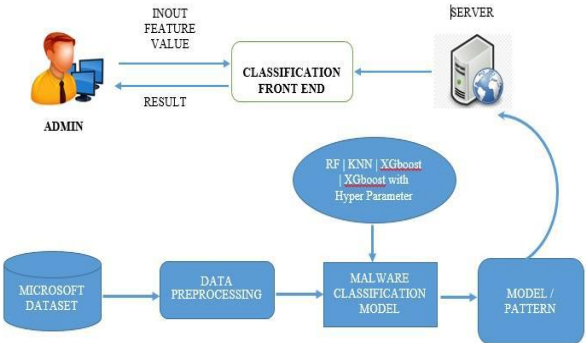

Fig 1: System Design

This system aims to develop a method which can classify variants of malware in a system with the help of Machine Learning algorithm using Microsoft Kaggle dataset with better predictive accuracy. Random forests are constructed by combining several trees with isolation training. Unlike boosting the base models by means of a sophisticated weighting system, the trees are typically trained independently and trees predictions combined by average. Every tree is developed separately in the XGBoost Classifier with best hyper parameter. We don't bootstrap between separate trees unlike the wild forest of Breiman (2001).

XGBoost algorithm is used to train the module using Cleveland collection of dataset instead of having one target class. Both the prior details was utilized by the current system to forecast the status of the problem and it will not take part in preparation and testing processes to isolate the specific data, rendering it ideal for knowledge updating because each new problem represents an improved dataset.

V. SOFTWARE REQUIREMENT SPECIFICATION

A. Software Requirements

- Operating System: Windows 8.1 Platform or Above
- Programming Language: Python 3.6.7
- Framework: Jupiter Notebook
- Cloud Platform: Google Cloud Engine (GCE)

B. Hardware Requirements

- Processor: Intel core i3 1.60GHz or above
- Hard Disk: 250 GB
- RAM: 4.00 GB
- Input: Keyboard and Mouse
- Output Device: High Resolution Monitor

C. Functional Requirements

- **Data pre-processing:** The purpose of pre-processing is to check for missing values in the dataset. If any such values are found, it is replaced by mode of the corresponding values.
- **Feature Extraction:** All 52 features of .asm file are input to the classifier. This module selects a subset from the actual classifier. This process is usually done to improve the accuracy and reduce the training time when the number of feature is very large.
- **Hyper tuning module:** It is here that the values of the parameters of the classifier are changed in order to increase the performance of classifier. The parameters can be varied and the one which gives the better accuracy is selected as the model.
- **Results:** Confusion Matrix, Log Loss.

D. Non Functional Requirements

- **PERFORMANCE REQUIREMENT**, low test log loss rate has been successfully achieved using XGBoost Classifier with Hyper Parameters for both .bytes and .asm files individualy, and as well as after merging features of .bytes and .asm files.
- **SOFTWARE QUALITY REQUIREMENT**, maximum possible accuracy has been achieved using XGBoost Classifier with hyperparameters using Random Search with log loss of 0.385, XGBoost Classifier with log loss of 0.0427, and Random Forest Classifier with log loss of 0.4192. It generated the confusion matrix. It used minimal resources for training the dataset as well as obtaining the results. The module is reliable and can be used to classify most of the malware in the validation set.

VI. DATA DESCRIPTION

Microsoft Kaggle Dataset is used. Microsoft provides the details on the website of Kaggle. The data size is 200 GB (uncompressed). 10,868 research samples are usable. The corresponding .asm and .bytes files for every sample are available. The files are generated using the disassembler device of IDA. 10,868 samples have been collected each of .bytes and .asm files.

Every sample belongs to one of the nine separate malware classes: Ramnit, Lollipop, Kelihos_ver3, Vundo, Simda, Tracur, Kelihos_ver1, Obfuscator.acy, Gatak.

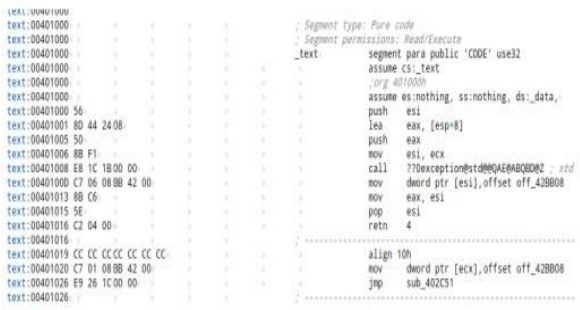

Fig 2: .asm file format

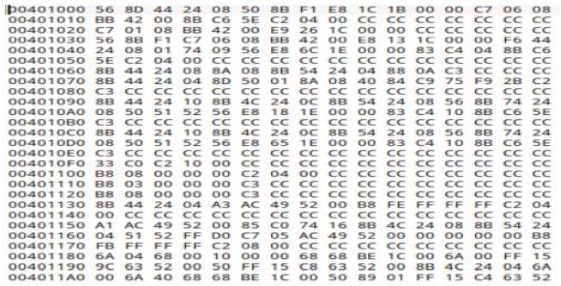

Fig 3: .bytes file format

VII. IMPLEMENTATION

A. Feature Extraction

- Based on the analysis of the data, it is found that .bytes and .asm file sizes would be useful in integrating them in the class family prediction.
- For .bytes files, only file sizes are used as the feature.
- For .asm files, we extracted 52 features from all the .asm files which are important.

B. Machine Learning Models on .bytes and .asm files

1. **KNN Classifier:** When KNN classifier was used, log loss of 0.4626 was obtained for .bytes file and log loss of 0.0798 was obtained for .asm file.
2. **Logistic Regression Classifier:** When Logistic Regression classifier was used, log loss of 0.8516 was obtained for .bytes file and log loss of 0.3992 was obtained for .asm file.
3. **Random Forest Classifier:** When Random Forest classifier was used, log loss of 0.0753 was obtained for .bytes file and log loss of 0.0323 was obtained for .asm file.
4. **XGBoost classifier:** When XGBoost classifier was used, log loss of 0.0654 was obtained for .bytes file and log loss of 0.0303 was obtained for .asm file.
5. **XGBoost Classification with best hyper parameters using Random Search:** In this case, log loss of 0.0624 was obtained for .bytes file and log loss of 0.0257 was obtained for .asm file.

Random Forest Classifier, XGBoost Classifier, and XGBoost Classifier with hyperparameters, after merging features of .bytes files and .asm files:

On applying the machine learning models on features of .bytes files and .asm files individually, the least log loss was achieved for XGBoost classifier with hyper parameter using Random Search, XGBoost classifier, and Random Forest classifier. After merging the features of .bytes and .asm files, we used Random Forest classifier, XGBoost classifier, and XGBoost classifier with hyper parameter using Random Search again on final features and we were able to achieve the log loss of 0.0419, 0.04275, and 0.0385 respectively.

VIII. FINAL RESULT

The table below provides an overview of the results of all our classification models. The Microsoft Kaggle for each of our models and the cross-validation performance for the training data are also discussed.

TABLE I. FINAL RESULT

MODEL	LOG LOSS (.bytes)	LOGLOSS (.asm)
KNN	0.4626	0.0798
LOGISTIC REGRESSION	0.8516	0.3992
RANDOM FOREST	0.0753	0.0323
XGBOOST CLASSIFIER	0.0654	0.0303
XGBOOST CLASSIFIER WITH HYPER PARAMETER	0.0624	0.0257

After merging the features of .bytes and .asm file, malware files were reclassified using Random Forest (RF) classifier, XGBoost classifier, and XGBoost classifier with hyper parameter using Random Search, and we were able to achieve the Log loss of 0.04192, 0.0427 and 0.0385 respectively.

IX. CONCLUSION

The classification of malwares was successfully done using machine learning algorithms on Microsoft Kaggle Dataset, and XGBoost Classifier with hyperparameters showed the least log loss.

The key difference between the models used and the top classifications was, instead of explicitly using the produced probabilities, to do more adjustment and tuning of the probabilities expected by the model. In terms of efficiency within our

respective implementations, the types of apps and models did not vary too greatly aside from the above discrepancy. The collection of features is essential for the model's efficiency. A study of the data provides valuable insights into this aspect. Redundant and noisy features could reduce the accuracy of classification. For example, log loss had increased when we tried to integrate "Call frequency system" with the existing models.

Interestingly, the trade-off between computer time and the accuracy of the model has come to be part of our project. It took approximately half a day to train the model on the entire data collection used, after using about 52 features and training a XGBoost classifier with hyper parameter using Random Search, we were able to achieve the Log loss of 0.0385, after merging the features of .bytes and .asm files.

X. FUTURE SCOPE

Although the classification features which were used in this project have been quite good, we can still try to incorporate more interesting features, particularly given the fact that most of the features used in this project are based on frequency. For example, research [11] shows how malware files can be interpreted to support the task of classification. So, we could build such "pixels" as.asm and .bytes files features. They could help to differentiate the families of malware. In fact, if we use real network call frequency calls as options rather than just frequencies, it may be worth testing if the network call frequency function used contributes to a stronger log loss. This can be achieved by constructing a bit vector for all system calls and setting the corresponding fractions for system calls invoked in the file (there are just around 50 system calls).

Then, as a feature, we can use each bit. Several of the top graders who claim it is an excellent function have used the header detail. This is a very interesting feature, because it only examines a small part of the data to be classified. Thus, this task of intelligent functional selection is an interesting and difficult one, in which there is much to learn.

REFERENCES

[1]. The data set is downloaded from the following link http://www.kaggle.com/c/malware-classification
[2]. Base paper downloaded from the kaggle website: Rakesh Chada, Nitish Gupta, "Microsoft Malware Classification Challenge".
[3]. Royi Ronen, Marian Radu, Corina Feuerstein, Elad Yom-Tov, and Mansour Ahmadi (2018), "Microsoft Malware Classification Challenge".
[4]. https://towardsdatascience.com/malware-classification-using-machine-learning-7c648fb1da79
[5]. https://www.researchgate.net/figure/Malware-families-in-the-dataset_tbl1_323470001
[6]. Dahl, George E., et al."Large-scale malware classification using random projections and neural networks." Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on. IEEE, 2013.
[7]. Annachhatre, Chinmayee, Thomas H. Austin, and Mark Stamp. "Hidden Markov models for malware classification." Journal of Computer Virology and Hacking Techniques (2014): 1-15.
[8]. Weber, Michael, et al."A toolkit for detecting and analyzing malicious software."Computer Security Applications Conference, 2002. Proceedings. 18th Annual. IEEE, 2002.
[9]. https://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Bilar.pdf
[10]. Austin, T. H., Filiol, E., Josse, and Stamp, S. M. ˆaAIJExploring Hidden Markov Models for Virus ˇ Analysis: A Semantic Approach, Proceedings of the 46th Hawaii International Conference on System Sciences, Wailea, HI, USA, 2013, Jan 7-10, 50395048
[11]. Nataraj, Lakshmanan, et al." Malware images: visualization and automatic classification." Proceedings of the 8th international symposium on visualization for cyber security. ACM, 2011.
[12]. https://www.kaggle.com/c/malware-classification/forums/t/13509/brief-description-of-7th-place-solution/72485
[13]. J.O. Kephart and W.C. Arnold, "Automatic Extraction of Computer Virus Signatures," Proc. Fourth Virus Bull. Int'l. Conf., pp. 178-184, 1994.
[14]. K. Griffin, S. Schneider, X. Hu, and T. Chiueh, "Automatic Generation of String Signatures for Malware Detection," Proc. 12th Int'l Symp. Recent Advances in Intrusion Detection, 2009.n