Static PE Malware Detection using Gradient Boosting Decision Trees algorithm

Huu-Danh Pham[1] and Thanh-Nguyen Vu[1]

Vietnam National University Ho Chi Minh City,
University of Information Technology, Ho Chi Minh City, Vietnam
{14520134,nguyenvt}@uit.edu.vn

Abstract. Static malware detection is an essential layer in a security suite, which attempts to classify samples as malicious or benign before execution. However, most of the related works incur the scalability issues, for examples, methods using neural networks usually take a lot of training time [13], or use imbalanced datasets [17,20], which makes validation metrics misleading in reality. In this study, we apply a static malware detection method by Portable Executable analysis and Gradient Boosting Decision Tree algorithm. We manage to reduce the training time by appropriately reducing the feature dimension. The experiment results show that our proposed method can achieve up to 99.394% detection rate at 1% false alarm rate, and score results in less than 0.1% false alarm rate at a detection rate 97.572%, based on more than 600,000 training and 200,000 testing samples from Endgame Malware BEnchmark for Research (EMBER) dataset [1].

Keywords: Malware detection · Machine learning · PE file format · Gradient boosting decision trees · EMBER dataset.

1 Introduction

Malware is typically used as a catch-all term to refer to any software designed to cause damage to a single computer, server, or computer network. A single incident of malware can cause millions of dollars in damage, i.e., zero-day ransomware WannaCry has caused world-wide catastrophe, from knocking U.K. National Health Service hospitals offline to shutting down a Honda Motor Company in Japan [3]. Furthermore, malware is becoming more sophisticated and more varied everyday. Accordingly, the detection of malicious software is an essential problem in cybersecurity, especially as more of society becomes dependent on computing systems.

Malware detection methods can be classified in either static malware detection or dynamic malware detection [7]. Static malware detection classifies samples as malicious or benign without executing them, in contrast to dynamic malware detection which detects malware based on its runtime behavior. In theory, dynamic malware detection provides the direct view of malware action, is less vulnerable to obfuscation, and makes it harder to reuse existing malware

[11]. But in reality, it is hard to collect a dataset of malware behavior because the malwares can identify the sandbox environment and prevent itself from performing the malicious behavior. In contrast, although static malware detection is known to be undecidable in general [5], it has enormous datasets which can be created by aggregating the binaries files, and is a critical layer in a security suite because when successful, it allows identifying malicious files before execution.

In this study, we present and optimize a static malware detection method using hand-crafted features derived from parsing the PE files and Gradient Boosting Decision Trees (GBDT), a widely-used powerful machine learning algorithm. Rather than using raw binary files, our proposed method use the statistical summaries to decrease the privacy concerns of various benign files and makes it easy to request the balanced dataset. GBDT also shows its efficiency and accuracy by taking less time for training and achieving impressive evaluation results.

2 Related Work

Malware detection has grown over the past several years, due to the more rising threat posed by malware to large businesses and governmental agencies. Various machine learning-based static Portable Executable (PE) malware detection methods have been proposed since at least 1995 [9,18,10,17,13]. In 2001, Schultz et al. represented PE files by features that included imported functions, strings, and byte sequences [18]. In 2006, Kolter et al. used byte-level N-grams and techniques from natural language processing, including TFIDF weighting of strings to detect and classify malicious files [10]. In 2015, Saxe and Berlin leveraged novel by using a histogram of byte entropy values for input features and a multilayer neural network for classification [17]. In 2017, Edward Raff et al. showed that fully connected and recurrent networks could be applied in the malware detection problem [14]. They also extends those results by training end-to-end deep learning networks on entire, several million byte long executables, and encounters a wide breadth of potential byte content [13].

In Vietnam, Nguyen Van Nhuong et al. proposed a semantic set method to detect metamorphic malware in CISIM 2015 [19]. Vu Thanh Nguyen et al. proposed a combined method of Negative Selection Algorithm and Artificial Immune Network for virus detection in in FDSE 2014 [12], and a metamorphic malware detection system by Portable Executable Analysis with the Longest Common Sequence in FDSE 2017 [20].

3 Proposed Method

3.1 The Issues of using Imbalanced Dataset

Most of the related work use the imbalanced dataset [17,20]. For examples, Saxe and Berlin used the dataset of 431,926 binaries which consists 350,016 malicious files [17], Vu Thanh Nguyen et al. used the dataset of 9690 files which only has 300 benign files [20].

In fact, the number of malicious files is often much more massive than the number of benign files because almost benign binaries are often protected by the copyright laws which do not allow for sharing. This makes malware identification problem become different from other machine learning classification problems, which commonly have fewer samples in important classes. Further, the size of the dataset is usually not large enough because the malware analysis and data labeling are consuming processes that required well-trained security engineers. There are also many risks in publishing a large dataset that includes malicious binaries.

Using imbalanced datasets can make validation metrics misleading. For examples, with 96.9% of data is malicious files, a model that labels all samples as malware achieves 96.9% accuracy, 96.9% precision (P), 100% recall (R) and 0.9843 F-score ($F = 2PR/(P + R)$ [4]). It also give way to false positives, which cause negative user experiences. According to a survey of IT administrators in 2017, 42 percent of companies assume that their users lost productivity as an issue of false-positive results, which creates a choke point for IT administrators in the business life cycle [6].

3.2 Feature Extraction

By using the simple feature extraction methods inspired by the EMBER dataset owners rather than raw binary files, collecting data is not affected by privacy policies and its is much more easier to get a balanced dataset. By conducting many experiments, we decrease the feature dimension by 30 percent (1711 instead of 2351) to reduce the training time but still manage to achieve a better evaluation results. In detail, we extracted each Portable Executable file into eight feature groups which can be classified into two types: format-agnostic features (byte-entropy histogram, byte histogram and string information) and parsed PE features. File-format agnostic feature groups decrease privacy concerns while parsed PE feature groups encapsulates the information related to executable code.

Byte-Entropy Histogram The work [17] shows that, in practice, representing byte values in the entropy context in which they occur separates byte values from the context effectively. For example, x86 instruction data from byte values occurring in compressed data. To compute the byte-entropy histogram, we slide a 2048-length window over all the input bytes with a step size of 1024 bytes. Use a simple trick to calculate the entropy H faster, i.e., reducing the information by half, and pairing it with each byte within the window. Then, we compute a two-dimensional histogram with 16×16 bins that quantize entropy and the byte value. Finally, row vectors in the matrix are concatenated and normalized the final 256-value vector.

Byte Histogram The byte histogram is a 256-value vector which represents the distribution of each byte value within the file.

String Information The final format-agnostic group of features is string information. These features are derived from the printable sequences of characters in the range `0x20` to `0x7f`, that have at least five characters long. We use the number of strings, the average length of these strings, the amounts of lines that may sequentially indicate a path (begin with `C:\`), an URL(start with `http://` or `https://`), a registry key (the occurrences of `HKEY_`) and a bundled executable (the short string `MZ`). Also, we use a histogram of the printable characters within these strings.

Parsed PE Features Five other feature groups are delivered from parsing the Portable Executable file format. The first group is the general information group, which includes the file size and necessary information collected from the PE header (the virtual size of the file, the number of imported and exported functions, the number of symbols, whether the data has a debug section, thread local storage, resources, relocations, or a signature). The second group is the information from Common Object File Format header (the timestamp in the header, the target machine and a list of image characteristics) and from the optional header (target subsystem, DLL characteristics, the file magic as a string, major and minor image versions, linker versions, system versions and subsystem versions, and the code size, header size and commit size). The next two feature groups are imported and exported functions. we apply the 512-bin hashing trick to capture individual imported functions, by representing it in string format `library:function`, for example, `kernel32.dll:CreateFileMappingA`. Simarly, a list of the exported functions is hashed into a list of 128-value vectors. The last group contains section information, i.e., the name, size, entropy, virtual size, and a list of strings representing section characteristics.

3.3 Gradient Boosting Decision Trees

For classification, we propose the method which uses the traditional Gradient Boosting Decision Trees (GBDT) algorithm with 400 iterations and 64 leaves in one tree. In the training step, we configure that there must be at least 200 samples in one child, and set learning rate at 5 percent. The reasons for choosing GBDT are below.

Scalability There are many essential features that is need to classify malicious and benign files. Even after feature selection, the massive number of features causes scalability issues for many machine learning algorithms. For example, non-linear SVM kernels require $O(N^2)$ multiplication during each iteration, and k-Nearest Neighbors (k-NN) requires significant computation and storage of all label samples during prediction. Accordingly, scalable alternatives are used in malware detection, and the two most popular algorithms are neural networks and ensemble decision trees.

Training time One of the most challenges that anti-malware systems usually have is the enormous amounts of data which needs to be evaluated for possible malicious intent [16]. Those results will be used to train the model to automatically evaluate samples in future. Hence, the training time becomes the essential criteria. Using neural networks usually take a long time for training. For examples, MalConv, the end-to-end deep learning model [13], took 25 hours for each epochs in training with the binaries from EMBER dataset [1]. In contract, tree ensemble algorithms handle very well a large number of training samples in moderate training times. One of popular tree ensemble algorithms, which has shown its effectiveness in several challenges, is gradient boosting decision tree. GBDT is widely-used and achieves state-of-the-art performances in many tasks, such as ranking [2] and click prediction [15]. Besides, Guolin Ke et al. recently release LightGBM, which is a highly efficient gradient boosting framework that speeds up the training process up to over 20 times while achieving almost the same accuracy [8].

4 Experiment

4.1 Dataset

In our experiment, we use 600,000 labeled training samples and 200,000 testing samples from Endgame Malware BEnchmark for Research (EMBER) dataset [1].

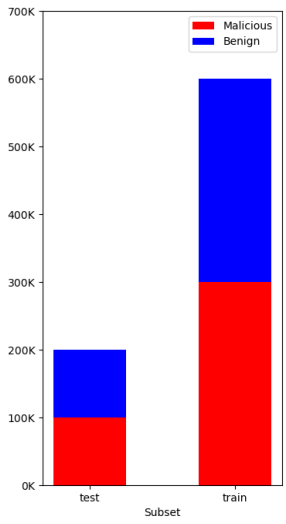

Fig. 1. Distribution of samples in EMBER dataset.

4.2 Evaluation Criteria

False Alarm Rate False positives, or false alarms, happen when a detector mistakes a malicious label for a benign file. We intend to make the false positive rate as low as possible, which is untypical for machine learning application. It is important because even one false alarm in a thousand benign files can create severe consequences for users. This problem is complicated by the fact that there are lots of clean files in the world, they keep appearing, and it is more challenging to collect these files. We evaluate the accuracy of our method at two specific false alarm rate values: at less than 0.1%, and at less than 1%.

$$False\ alarm\ rate = \frac{\sum False\ positive}{\sum Condition\ negative}$$

Detection Rate The detection rate, (eqv. with recall or true positive rate), measures the ratio of malicious programs detected out of the malware files used for testing. With higher recall, fewer actual cases of malware go undetected.

$$Detection\ rate = \frac{\sum True\ positive}{\sum Condition\ positive}$$

Area Under the ROC curve The Area Under the ROC curve, AUROC or AUC for short, provides an aggregate measure of performance across all possible classification thresholds. AUC is scale-invariant and measures how well predictions are ranked, rather than their absolute values. Besides, AUC is classification-threshold-invariant, so that it can measure the quality of the predictions irrespective of what threshold is chosen. A model whose predictions are 100% wrong has an AUC of 0.0, and the one whose predictions are 100% correct has an AUC of 1.0.

4.3 Experimental Results

The proposed GBDT-based malware detection method is implemented with LightGBM framework [8], and the input feature vectors have dimension of 1711. All our experiments were run on an instance which has 24 vCPUs and 32 GB memory. Using parallel programming, it took about 10 minutes to vectorize the raw features and about 5 minutes to train the model. The ROC curve of the final model is shown in Figure 2.

The area under ROC curve exceeds 0.999661, which means that almost all the predictions are correct. With a threshold of 0.828987, the model score results in less than 0.1% false alarm rate at a detection rate 97.5720%. At less than 1% false positive rate, the model exceeds 99.3940% detection rate with a threshold of 0.307897.

The baseline model has only the area under the ROC curve of 0.999678, score results in less than 0.1% FPR at TPR exceeding 92.99%, and at less than 1% FPR, it exceeds 98.2% TPR. Our model has better performance because of

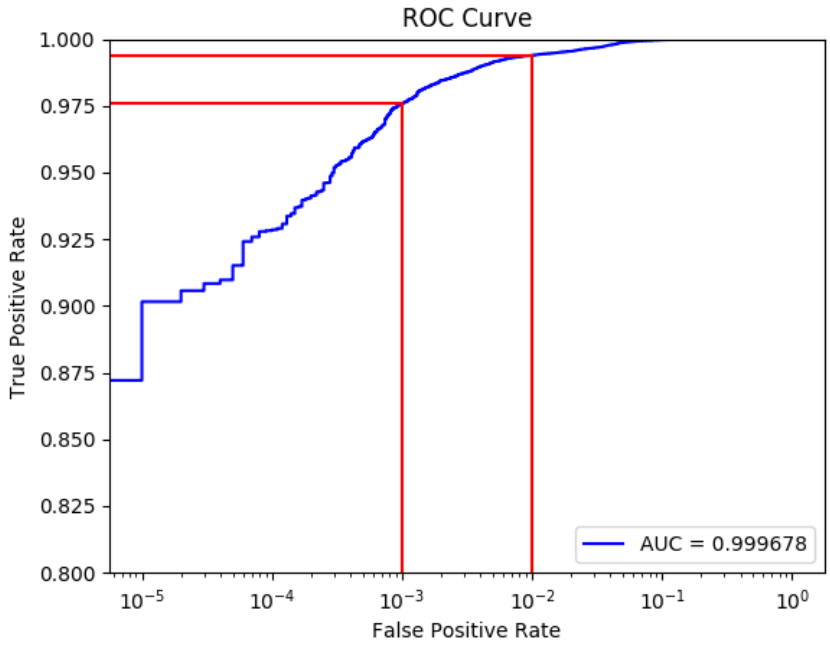

Fig. 2. The ROC curve of proposed model

hyper-parameter tuning, and it also takes less time for training as a result of reducing feature space. Evidently, the model has better performance than the MalConv model trained on the raw binaries [1], which has ROC AUC is 0.99821, corresponding to a 92.2% TPR at a FPR less than 0.1%, and a 97.3% TPR at a less than 1% FPR.

5 Conclusion and Future Works

In this study, we show the weakness in earlier proposed methods, especially the issues in using imbalanced datasets in malware detection. We demonstrate feature extraction using file-format agnostic features decreases privacy policies and makes it easier to collect benign files. We have also applied a method for static PE malware detection using gradient boosting decision trees algorithm. On the EMBER dataset, our proposed model achieve a better performance than the baseline model introduced by the dataset owners while takes less training time by reducing the feature vector dimension. Besides, the validation results reinforce the evidence that leveraging domain knowledge from parsing the PE format file has still achieved a remarkable performance.

The study conducted in this project was a proof-of-concept, and we can identify some future developments related to the practical implementation.

Reduce the feature space There is possible to reduce the dimension of feature vectors. Input vectors with smaller size boost the model and take less training time.

Use other datasets Although the EMBER dataset is broad, covering most of the malware species, it does not include all possible kinds. Collecting a dataset is a task that requires a lot of time and efforts, especially in malware detection domain. With using format-agnostic features, we can receive more samples from security organizations in future.

References

1. Anderson, H.S., Roth, P.: Ember: An open dataset for training static pe malware machine learning models. arXiv preprint arXiv:1804.04637 (2018)
2. Burges, C.J.: From ranknet to lambdarank to lambdamart: An overview. Tech. rep. (June 2010)
3. Chen, Q., Bridges, R.A.: Automated behavioral analysis of malware A case study of wannacry ransomware. CoRR (2017)
4. Chinchor, N.: Muc-4 evaluation metrics, in proceedings of the fourth message understanding conference (muc-4). Morgan Kaufman Publishers p. 22 (1992)
5. Cohen, F.: Computer viruses: theory and experiments. Computers & security **6**(1), 22–35 (1987)
6. Crowe, J.: Security false positives cost companies $1.37 million a year on average (2017)
7. Egele, M., Scholte, T., Kirda, E., Kruegel, C.: A survey on automated dynamic malware-analysis techniques and tools. ACM computing surveys (CSUR) **44**(2), 6 (2012)
8. Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., Liu, T.Y.: Lightgbm: A highly efficient gradient boosting decision tree. In: Advances in Neural Information Processing Systems. pp. 3146–3154 (2017)
9. Kephart, J.O., Sorkin, G.B., Arnold, W.C., Chess, D.M., Tesauro, G.J., White, S.R., Watson, T.: Biologically inspired defenses against computer viruses. In: IJCAI (1). pp. 985–996 (1995)
10. Kolter, J.Z., Maloof, M.A.: Learning to detect and classify malicious executables in the wild. Journal of Machine Learning Research **7**(Dec), 2721–2744 (2006)
11. Moser, A., Kruegel, C., Kirda, E.: Limits of static analysis for malware detection. In: Computer security applications conference, 2007. ACSAC 2007. Twenty-third annual. pp. 421–430. IEEE (2007)
12. Nguyen, V.T., Nguyen, T.T., Mai, K.T., Le, T.D.: A combination of negative selection algorithm and artificial immune network for virus detection. In: Future data and security engineering, pp. 97–106. Springer (2014)
13. Raff, E., Barker, J., Sylvester, J., Brandon, R., Catanzaro, B., Nicholas, C.: Malware detection by eating a whole exe. arXiv preprint arXiv:1710.09435 (2017)
14. Raff, E., Sylvester, J., Nicholas, C.: Learning the pe header, malware detection with minimal domain knowledge. In: Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security. pp. 121–132. ACM (2017)
15. Richardson, M., Dominowska, E., Ragno, R.: Predicting clicks: estimating the click-through rate for new ads. In: Proceedings of the 16th international conference on World Wide Web. pp. 521–530. ACM (2007)

16. Ronen, R., Radu, M., Feuerstein, C., Yom-Tov, E., Ahmadi, M.: Microsoft malware classification challenge. arXiv preprint arXiv:1802.10135 (2018)
17. Saxe, J., Berlin, K.: Deep neural network based malware detection using two dimensional binary program features. In: 2015 10th International Conference on Malicious and Unwanted Software (MALWARE). pp. 11–20 (2015). https://doi.org/10.1109/MALWARE.2015.7413680
18. Schultz, M.G., Eskin, E., Zadok, E., Stolfo, S.J.: Data mining methods for detection of new malicious executables. In: Proceedings of the 2001 IEEE Symposium on Security and Privacy. pp. 38– (2001)
19. Van Nhuong, N., Nhi, V.T.Y., Cam, N.T., Phu, M.X., Tan, C.D.: Semantic set analysis for malware detection. In: IFIP International Conference on Computer Information Systems and Industrial Management. pp. 688–700. Springer (2014)
20. Vu, T.N., Nguyen, T.T., Trung, H.P., Do Duy, T., Van, K.H., Le, T.D.: Metamorphic malware detection by pe analysis with the longest common sequence. In: International Conference on Future Data and Security Engineering. pp. 262–272. Springer (2017)