A Hybrid Positive-Unlabeled Learning Method for Malware Variants Detection

Alle Giridhar Reddy(✉), Pushkar Kishore, Swadhin Kumar Barisal, and Durga Prasad Mohapatra

Department of Computer Science, National Institute of Technology Rourkela, Rourkela, Odisha, India
{518CS1002,durga}@nitrkl.ac.in

Abstract. Malware are capable of evolving into different variants and conceal existing detection techniques, which relinquishes the ineffectiveness of traditional signature-based detectors. There are many advanced malware detection techniques based on machine learning and deep learning, but they cannot fulfill the real issues in industries. Malware variants are evolving at a rapid pace and labelling each of them is not practical and feasible. So, industries are considering a lot of the unlabeled samples as benign, while only a few are labelled. Consequently, the authentic malware samples are mislabelled. Bias created by mislabelling the samples severely restraints the accuracy. Also, the user is unsatisfied with malware detection system, since there is poor negotiation between the speed and accuracy.

In this research article, we propose a hybrid positive-unlabeled learning technique for malware detection that can address some important challenges. Here, we use an ensemble model comprising of Logistic regression (cost-sensitive boosted), Random Forest and Support vector machine, to detect the malware variants. Along with that, we demonstrate that features in the form of a triplet vector are optimal while training a model. Experimental outcomes show that our proposed model attains 91% malware detection accuracy having a false alarm rate less than 0.005, while the earlier state-of-art approaches can only achieve 76.4% to 89% accuracy. The detection speed of our approach is 0.003 s.

Keywords: Ensemble model · Positive-unlabeled learning · Machine learning techniques · Malware

1 Introduction

Malware is an extensive threat that covers computers as well as Internet of Things (IOT) devices [1]. Whenever software under study has coding or configuration error and wrongly sensed as anomalous, it generates a false positive data [7,11,14]. For the machine learning (ML) or deep learning (DL) based techniques to work, we must label several legitimate and benign binary executables for training. Labeling each sample is inimical as it takes a lot of time and requires

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2021

high labour costs. For this reason, the volume of the unlabeled sample is smaller compared to labeled one. For tackling this issue, companies label the remaining unlabeled malware samples as benign.

Now, the unlabeled binary executable is considered as benign one (negative samples), but the authentic malicious binary executable (positive samples) are mislabeled. Immense amount of malicious samples in the unlabeled binary executable will reduce the efficiency of the malware detection model. So, it generates bias in the decision boundary and is illustrated in Fig. 1, which leads to an inaccurate malware detector. We term this problem as positive-unlabeled learning, where a 2-class classifier is trained using a dataset containing negative data (benign), positive data (malicious) and unlabeled data (presumed as benign).

In this research work, our primary goal is to enhance the accuracy of the detector when it is trained using the dataset created from positive-unlabeled binary executable. Besides, detecting malware variants, we use system calls which is collected using NITRSCT [2] tool and demonstrate that the vector of consecutive 3 system calls as a feature, will be effective and optimal for malware detection. Whenever a dataset is created, we optimize the logistic regression with cost-sensitive boosting. We propose an ensemble model comprising optimized logistic regression, random forest and support vector machine, which will accurately detect malware variants and will have least detection time.

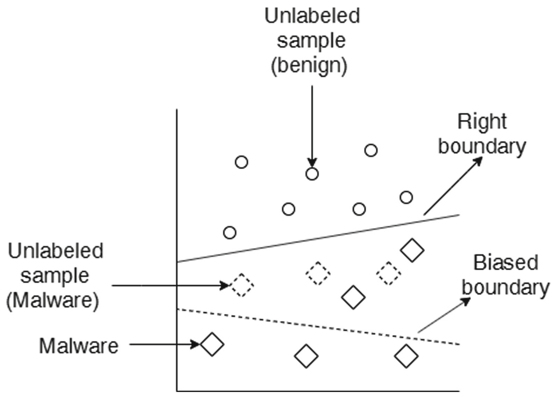

Fig. 1. Biased boundary misclassifies the samples [20]

We organize the remaining part of this article as follows. Section 2 consists of related works, Sect. 3 presents our proposed malware detector's methodology. Section 4 exhibits the experimental results. Section 5 presents the comparison with related work. Section 6 discusses over the threats to validity and Sect. 7 concludes the paper with future directions.

2 Related Works

In this section, we discuss some existing work linked to our approach.

2.1 Malware Variants Detection

Fan et al. [3] recommended subgraphs construction of API calls which describes the prevalent behaviour of binary executable malware of the same family. But, the extraction of API calls from binary executable fails sometimes. Zhang et al. [4] embedded opcode and API sequential calls using Convolutional neural networks (CNN) and Back Propagation Neural Networks (BPNN) and the mix of these networks is used for training a malware variant detector having hybrid features. Zhang et al. [5] used topological features of the opcode graph to identify android binary executable malware. Stringhiniq et al. [8] created a graph depicting the file delivery networks and trained using semi-supervised Bayesian label propagation. Here, reputation of the acknowledged files is sent to all nodes of the graph. But these files are behind-time, meaning that whenever malicious binary executable is sensed, then up to that time, countless copies of the malware has already done the damage.

Canzanese et al. [6] represented binary executable using system call n-gram and used SVM for malware variants detection. Raff et al. [19] used CNN and bytecode n-grams for malware variants detection. Presence of noise in bytecodes is much higher compared with opcodes, resulting in a depreciation of accuracy. Kang et al. [18] used Naive Bayes technique towards catching the 2-opcode vectors of binary malicious executable. But, the assumption that Naive Bayes considers features as independent, depreciates the malware detector's accuracy. Puerta et al. [10] used SVM for detecting malware and represented binary executable using opcode frequencies. Simplicity of features depreciates the accuracy due to the lack of the adequate information in the features.

All the above discussed approaches rely on dataset having known positive and negative labels. If they will design detector with dataset having semi-labeled instances, then the accuracy is acutely hampered.

2.2 Positive-Unlabeled Learning

Some scholars preferred positive-unlabeled learning techniques for training the malware detector whenever they encounter positive-unlabeled data. Liu et al. [12,13] designed a SVM with some bias and upcoming steps were used to find obvious positive samples. Xiao et al. [16] employed K-means to discover positive as well as negative instances in dataset which are unlabeled. But, these techniques were influenced by the recognized negative training samples. Malware detection results' accuracy will be disastrous if the negative data is inaccurately detected. Elkan et al. [17] trained a malware detection classifier on positive as well as unlabeled instances for estimating weights of validation set samples and modeled a weighted Support Vector Machine. Xu et al. [9] generalized solitary positive class of binary executable into various positive classes. Gong et al. [15]

identified the margin among positive and potentially negative samples and proposed a label-calibrated SVM. However, their method is confined to SVM and is inappropriate for other ML methods. Zhang et al. [20] suggested a cost-profound boosting technique for positive-unlabeled learning for malware detection. But they have used opcode representation, which is only the static analysis of the binary executable.

3 Proposed Methodology

In this section, we propose the methodology for detecting malicious variants using system calls and ML. First, we discuss the architecture of our proposed model. Second, we discuss the representation of system calls and conclude by presenting the positive unlabeled learning, which we use in our approach.

3.1 Architecture of Our Approach

We present the architecture of our malware detection model in Fig. 2. First, a model is initialized and trained using a customized cross entropy loss function and termed as optimized Logistic Regression. We use the cross-entropy function as it will output the binary label in the terms of probability. Then the training is done using the optimised LR, Support Vector Machine and Random Forest. After training is completed by the three different approaches, we ensemble them and use the hard voting classifier to predict the final label on the test dataset. In Ensemble model first, we will sum the predictions made by each model and then we will predict the class label with the most votes.

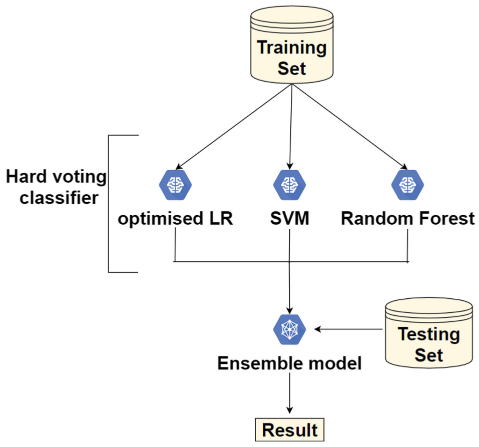

Fig. 2. Proposed architecture of our approach

3.2 Representation of System Calls

Features are extracted using n-gram model representing an adjacent sequence of n consecutive system calls derived from a selected sequential system call. We analyze three ML models' true positive rate (tpr), by varying the length of the vector of system calls. The tpr of each detector is shown by fixing the false positive rate at 10^{-5}. For n = 1, the highest tpr is 0.32, while for n = 2, the highest tpr is 0.85. When we select n = 3, the highest tpr is 0.91, and it decreases to 0.81 whenever n = 4 is considered. So, we can infer that taking the value of n = 3 will be optimal for the length of all feature's vector. Here, n is the number of consecutive system calls in a vector.

3.3 Positive-Unlabeled Learning

Nowadays, industries mark a limited portion of the binary malicious executable as positive sample and leftovers are "unlabeled legitimate data" containing malware variants. Upon training malware detector with "semi-labeled data" (unlabeled samples along with malware samples) using ML, the detection accuracy will be inadequate. In this article, we propose an ensemble method comprising of logistic regression (cost-sensitive boosted), random forest and SVM, that can boost the accuracy of positive unlabeled learning.

First, we optimize the logistic regression with cost-sensitive boosting. The idea behind this technique is assigning separate weights to unlabeled binary executable for differentiating between true legitimate binary executable and false legitimate binary executable (malware variants) in it. We calculate loss function using Eq. 1, where $Loss_{CEL}$ is the Cross Entropy Loss, h(x) is the confidence of x (model's yield) and y_i is the label of a sample in the tth iteration.

$$Loss_{CEL} = -(\sum_{i=0}^{n} \frac{h(x_i)}{\sum_{j=0}^{i-1} h(x_j)}.(y_i.log(h(x_i)) + (1-y_i).log(1-h(x_i)))) \quad (1)$$

The major problem is allocating proper weights to the samples. While training during each iteration, we allocate the weights bestowing to the confidence h(x). Let $\mathbf{x}$ be an instance of a dataset of binary executable, $\mathbf{x_m}$ is the malware instance and $\mathbf{x_b}$ is the benign one. Since $\mathbf{x_m}$ in unlabeled datasets and malware datasets are analogous, $\mathbf{h(x_m)}$ hovers between 0 and 1, while $\mathbf{h(x_b)}$ of x_b in unlabeled data sets will advance towards 1. Accordingly, $\mathbf{h(x)}$ of benign will be huge than $\mathbf{h(x)}$ of malicious binary executables. This amplifies the value of genuine legitimate binary executable and dampens the cost of false legitimate binary executable, while training with unlabeled data. The algorithm for our proposed hybrid positive-unlabeled learning malware variants detector is presented in Algorithm 1.

4 Experimental Results

We present the experiment for demonstrating that our proposed ensemble model can boost the accuracy with commendable detection time. At first, we present

Algorithm 1: Hybrid positive-unlabeled learning malware variants detection model

```
   Input: A data set $\mathbf{X}$
   Output: A detection model which classifies samples into malware and benign.
1  Initialize a model ML(x) ;
2  Set $avg = 0$ as initialization ;
3  Function Training($\boldsymbol{X}$, $ML(x)$):
4      for $i = 0 : n$ do
5          Train x using ML(x) ;
6          calculate h(x) from ML(x) ;
7          Get avg = (avg · i + h(x)) / (i + 1) ;
8          Update ML(x) according to Equation 1 ;
9      Apply Random Forest and SVM on $\mathbf{X}$ ;
10     ML(x) = Ensemble ML(x) (Optimized Logistic regression), Random forest
         and SVM using hard voting ;
11     return ML(x) ;
12 end function
```

the experimental setup, the dataset, the hyper-parameter settings and at last we conclude with performance evaluations.

4.1 Setup, Dataset and Hyper-parameters

We carry out entire experiments on one system. The variant of the CPU is Intel i5-3470 @ 3.20 GHz, the RAM is 16.0 GB and the OS is Windows 10. We have implemented our approach using Python language in which the matrix computations are dependent on numpy.

The dataset[1] considered for performance evaluation and training is developed using NITRSCT[2], which was developed by us. We evaluate the performance of our proposed model during training with the unlabeled datasets, which contain 20% malware variants and 80% benign ones.

For preparing a dataset, we have collected benign executables from 20 hosts in offices, laboratories and isolated testbeds. The malware used for experimental purpose is collected from VirusTotal[3]. The hyper-parameters which are fixed by us have a considerable effect on the performance. We display the hyper-parameters used in our way in Table 1.

4.2 Performance Analysis of Malware Detection

The parameters which we use for performance analysis of our proposed model are classification accuracy, detection false positive rate, detection true negative

[1] https://github.com/pushkarkishore/NITRSCT/blob/master/data1.rar.
[2] https://github.com/pushkarkishore/NITRSCT/blob/master/Debug.rar.
[3] https://www.virustotal.com/.

Table 1. The hyper-parameter settings of our experiment

Detector	Hyper-parameter	Value
Optimized Logistic Regression (OLR)	Learning rate	0.001
Optimized Logistic Regression (OLR)	Number of iterations	5000
Random Forest (RF)	Number of estimators	100
Random Forest (RF)	Random State	0
Random Forest (RF)	Number of Jobs	2
Support Vector Machine (SVM)	Kernel	Sigmoid
Proposed model	Weight	SVM(1), OLR(1), RF(1)

rate, detection false negative rate, detection precision, detection recall, F1-score, training time cost, detection time cost and area under curve. The classification accuracy is evaluated using Eq. 2. The recall of the model is the true positive rate, where True Positive (TP) is the number of correctly classified malware samples and False Negative (FN) implies malware samples misclassified as the benign one. TNR is the true negative rate, where False Positive (FP) is the number of benign samples misclassified as malware binaries and True Negative (TN) is the number of benign samples which are correctly classified. FPR represents false positive rate, FNR represents false negative rate, Precision represents malware detector's precision and F1-score is computed using Precision and Recall.

$$accuracy = \frac{TP + TN}{TP + FN + TN + FP} \tag{2}$$

The performance evaluation of our model is presented in Table 2. The ROC curve of our model is shown in Fig. 3. The area under the receiver operating characteristics (ROC) curve of our proposed model is 0.94. The confusion matrix of our model is presented in Fig. 4. For the combination of 1000 benign and malicious testing instances, we get 995 True negatives, 5 False positives, 180 False negatives, 820 True positives. Our model's False positive rate is 0.005 which is considerably low as compared to other works. A malware detector is practically useful when it has low FPR.

5 Comparison with Related Work

We have compared the performance of our model with that of several state-of-art methods and shown in Table 3. By comparing with the other state-of-the-art methods, we observe that our approach significantly improves the classification accuracy, precision, the (1-FPR), F1-score and training time while retaining the detection speed. Accuracy is better than all the considered models, making it useful for industrial malware detection. Precision is 99.3%, which is higher than the precision evaluated by using the SVM method. Recall is 82.0%, which is lower than logistic (optimization) and CNN (optimization) methods. It means that 82.0% of the total relevant results are correctly classified by our proposed

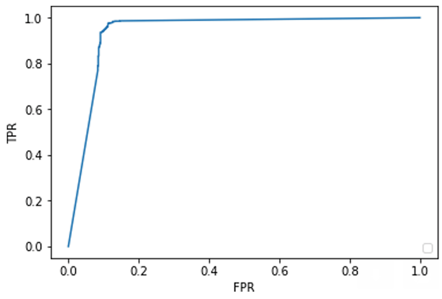

Fig. 3. Roc curve of our model

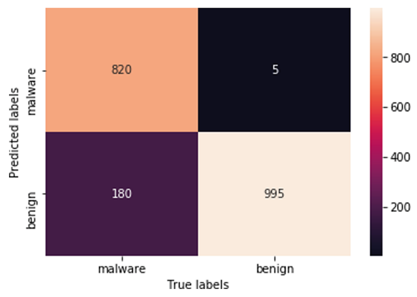

Fig. 4. Confusion matrix of our model

Table 2. Performance evaluation of our model

Sl. no.	Performance parameters	Value
1	Accuracy (%)	91.0
2	Recall (%)	82.0
3	TNR (%)	99.5
4	FPR (%)	0.5
5	FNR (%)	18.0
6	Precision (%)	99.3
7	F1-score (%)	89.6
8	Detection Time (s)	0.003
9	Training Time (s)	160

model. Considering the problem under consideration, we give the highest priority to either precision or recall. In general, we use a simple metric, F1-score, which is the harmonic mean of precision and recall. Our model proves its vitality when we consider F1-score, which is 89.6%. Specificity is equivalent to "1-FPR", which implies that benign samples being labeled benign is 99.5%. Its lower value will only block the benign process, so we consider it as an auxiliary parameter. Our extraneous objective of blocking of benign executable on the hosts will be minimal as specificity is higher. SVM [18] method has a higher precision, but it has lower F1-score, accuracy and recall. As accuracy and F1-score is lower than our proposed model, we cannot use this model. CNN (optimization) [20] method has higher recall than our proposed model, but lags in F1-score, accuracy and precision. Logistic (optimization) [20] method has also higher recall than our model, but has least accuracy, precision, specificity, F1-score, detection time as well as training time. Considering the above models, there is a poor trade-off between precision and recall, thus, their F1-score is minimal. Comparing with the above stated parameters, we observe that our proposed model is more suitable for malware detection.

Table 3. Comparison of performance of our approach with existing state-of-art approaches

Method	Accuracy	Precision	Recall	1-FPR	F1-score	Detection time	Training time
Logistic (optimization) [20]	89.0%	87.3%	91.2%	86.7%	89.2%	0.007 s	16,053.0 s
Softmax (optimization) [20]	80.0%	82.7%	75.8%	84.1%	79.1%	0.006 s	16,241.0 s
CNN (optimization) [20]	86.0%	86.2%	85.7%	86.3%	85.9%	0.053 s	107,519.0 s
CNN [19]	84.3%	91.8%	75.2%	93.3%	82.7%	0.053 s	109,633.0 s
SVM [18]	76.4%	92.9%	58.9%	95.5%	67.6%	0.006 s	676.0 s
Our approach	91.0%	99.3%	82.0%	99.5%	89.6%	0.003 s	160.0 s

6 Threats to Validity

For all the methods to work with system calls, they have to capture the system calls during runtime with the help of sandboxes, which makes it tough and costly in terms of resources in preparing dataset. Modern malware variants tend to hide their malicious behaviour whenever they detect themselves running in sandboxes. Our model can eliminate this issue to some level. We have analyzed every act of malware inactiveness or sleepy behaviour by making sandbox dynamically changing its time settings to deceive malware and stimulate its execution.

7 Conclusion and Future Work

In this paper, we proposed an ensemble malware detection method for positive-unlabeled learning adopted to detect numerous malware variants. In industries, malware detection model is trained with positive-unlabeled datasets, which severely limits the accuracy. Our approach addresses this issue by providing a novel ensemble malware variants detection model. Besides, we have demonstrated that a vector of three consecutive system calls, when considered as a feature in dataset will be optimal for malware detectors. The experimental results convey that our model achieves 91.0% accuracy with false alarm rate less than 0.005, while the other techniques achieve up to a maximum of 89%, when the unlabeled dataset contain many "mislabeled" data (positive data).

In the future, we can improve the cost of dataset creation and can detect newer sandbox-evading malware by enhancing the features of the sandboxes. We can consider a static analysis of the binary executable like API calls, opcode, etc. and design an ensemble detector to improve malware detection accuracy.

References

1. Meng, Y., Zhang, W., Zhu, H., Shen, X.S.: Securing consumer IoT in the smart home: architecture challenges and countermeasures. IEEE Wirel. Commun. **25**(6), 53–59 (2018)
2. Kishore, P., Barisal, S.K., Vaish, S.: NITRSCT: a software security tool for collection and analysis of kernel calls. In: IEEE Region 10 Conference (TENCON), pp. 510–515 (2019)

3. Fan, M., et al.: Android malware familial classification and representative sample selection via frequent subgraph analysis. IEEE Trans. Inf. Forensics Secur. **13**(8), 1890–1905 (2018)
4. Zhang, J., Qin, Z., Yin, H., Ou, L., Zhang, K.: A feature-hybrid malware variants detection using CNN based opcode embedding and BPNN based API embedding. Comput. Secur. **84**, 376–392 (2019)
5. Zhang, J., Qin, Z., Zhang, K., Yin, H., Zou, J.: Dalvik opcode graph based Android malware variants detection using global topology features. IEEE Access **6**, 51964–61974 (2018)
6. Canzanese, R., Mancoridis, S., Kam, M.: System call-based detection of malicious processes. In: Proceedings of IEEE International Conference on Software Quality, Reliability and Security (2015)
7. Barisal, S.K., Nayak, G., Naik, B.: Enhanced type safety in Java. Int. J. Comput. Appl. **47**(24), 12–16 (2012)
8. Stringhiniq, G., Shen, Y., Han, Y., Zhang, X.: Marmite: spreading malicious file reputation through download graphs. In: Proceedings of the 33rd Annual Computer Security Applications Conference (AC-SAC) (2017)
9. Xu, Y., Xu, S.C., Xu, C., Tao, D.: Multi-positive and unlabeled learning. In: International Joint Conferences on Artificial Intelligence, pp. 3182–3188 (2017)
10. De la Puerta, J.G., Sanz, B.: Using Dalvik opcodes for malware detection on Android. Logic J. IGPL **25**(6), 938–948 (2017)
11. Barisal, S.K., Behera, S.S., Godboley, S., Mohapatra, D.P.: Validating object-oriented software at design phase by achieving MC/DC. Int. J. Syst. Assur. Eng. Manag. **10**(4), 811–823 (2019)
12. Liu, B., Dai, Y., Li, X., Lee, W.S., Yu, P.S.: Building text classifiers using positive and unlabeled examples. In: IEEE International Conference on Data Mining, pp. 179–186 (2003)
13. Liu, B., Lee, W.S., Yu, P.S., Li, X.: Partially supervised classification of text documents. In: International Conference on Machine Learning, pp. 387–394 (2002)
14. Barisal, S.K., Dutta, A., Godboley, S., Sahoo, B., Mohapatra, D.P.: MC/DC guided test sequence prioritization using firefly algorithm. In: Evolutionary Intelligence, vol. 95, pp. 1–14. Springer (2019)
15. Gong, C., Liu, T., Yang, J., Tao, D.: Large-margin label-calibrated support vector machines for positive and unlabeled learning. IEEE Trans. Neural Netw. Learn. Syst. **30**, 3471–3483 (2019)
16. Xiao, Y., Liu, B., Yin, J., Cao, L., Zhang, C., Hao, Z.: Similarity-based approach for positive and unlabeled learning. In: International Joint Conferences on Artificial Intelligence, pp. 1577–1582 (2011)
17. Elkan, C., Noto, K.: Learning classifiers from only positive and unlabeled data. In: International Conferences on Knowledge Discovery and Data Mining, pp. 213–220 (2008)
18. Kang, B., Yerima, S.Y., McLaughlin, K., Sezer, S.: N-opcode analysis for android malware classification and categorization. In: Proceedings of International Conference On Cyber Security And Protection Of Digital Services (Cyber Security) (2016)
19. Raff, E., Barker, J., Sylvester, J., Brandon, R., Catanzaro, B., Nicholas C.: Malware Detection by Eating a Whole EXE, Proceedings of arXiv:1710.09435 (2017)
20. Zhang, J., Khan, M.F., Lin, X., Qin, Z.: An optimized positive-unlabeled learning method for detecting a large scale of malware variants. In: IEEE Conference on Dependable and Secure Computing (DSC), pp. 1–8 (2019)