Detecting Malware in Android Applications using XGBoost

Aleksandar Kitanovski, Hristina Mihajloska Trpcheska and Vesna Dimitrova
Faculty of Computer Science and Engineering
Ss. Cyril and Methodius University, Skopje, N. Macedonia

aleksandar.kitanovski@students.finki.ukim.mk,
hristina@finki.ukim.mk,
vesna.dimitrova@finki.ukim.mk

***Abstract*—The omnipresence of Android devices and the amount of sensitive information kept in them makes detecting malware in Android applications crucial. In this paper, the efficacy of using machine learning models for the purpose of malware detection in Android applications was examined, and several XGBoost models were developed and compared - each with a distinct feature set. We used the f1 score, precision, recall, confusion matrices, and precision-recall curves to compare the models. Accuracy was not considered since we needed a balanced dataset. One of the models we developed, which used all the available features in the dataset, had encouraging results with high precision and recall.**

***Index Terms*—XGBoost, detecting malware, Android applications**

I. Introduction

A. Malware Detection

Malware (or malicious software) is designed to harm intentionally, leak private information, gain unauthorized access, deprive users of access to information, or unknowingly interfere with the user's system security or privacy. Mobile malware is malware that explicitly targets mobile phones or personal digital assistants (PDAs). Common ways of avoiding malware are using antivirus software, and firewalls, applying regular patches to reduce the risk of zero-day attacks, creating regular backups, etc. But as mobile phones have become more common and complex, it has become increasingly challenging to ensure their safety from malware [5]. Also, malware is now being designed to avoid antivirus detection [13], and recent studies have shown that antivirus programs are ineffective against mobile malware due to its rapid evolution [12]; because of this, the development of models that can detect novel malware, without the need for manual rules or signatures, is crucial.

B. Classification

Classification is a process of categorizing a set of data into given classes and can be applied to many different fields, for example, spam detection, credit card fraud detection, medicine, finance, etc. Classification can be based on simple rules made by hand, or the classification function can be learned from a set of manually labeled data using machine learning methods. In this paper, classification based on machine learning for the detection of malware in Android applications was used.

C. Android Malware Dataset

The dataset used for training and evaluating the machine learning model used in this paper, is the Drebin-215 [14] dataset, which consists of 215 features and has 15 036 samples. 9476 of these are benign samples, while the other 5560 are malware samples from the Drebin project [1]. The features consist of API call signatures, Intents, and Manifest Permissions, which are extracted using static code analysis.

II. Related Work

Malware detection using machine learning is a hot topic, and there has been a lot of other work on detecting malware in Android applications using machine learning methods (both supervised and unsupervised). There are papers published about detecting malicious software using machine learning in computers [8], iOS devices [2], and even detecting malicious websites using natural language processing [4]. In the context of Android Malware Detection, on which our research focuses, researchers have been able to develop an explainable model based on support vector machines achieving a recall of 0.94 [1], others have used a one-class support vector machine to develop an anomaly detection model (which treats malware as an anomaly), achieving high recall but low precision [11]. A third group of researchers used a fusion of different models to create a model with high precision and high recall [15]. Graph Convolutional Networks [7] have also been successfully applied to the problem of detecting malware in Android applications, as well as Convolutional Neural Networks [10] achieving f1 scores of 0.986 and 0.97 respectively. A recently published study [9] suggested treating Android malware detection as a Bi-Level optimization problem where the upper-level optimization task aims to generate a set of detection rules, while the lower-level task, which is embedded in the upper-level task, aims to generate artificial malicious patterns which escape the rules of the upper-level. This method has also successfully achieved a precision of 0.9806 and recall of 0.9834. The XGBoost model has also been used in another research dealing with Android malware detection, achieving an accuracy of 0.946 [6].

III. Model

A. XGBoost

XGBoost stands for Extreme Gradient Boosting - it is a gradient-boosting model based on ensembles of decision trees. The term ”boosting” comes from the idea of boosting or improving a single weak model by combining it with a number of other weak models. XGBoost uses Gradient Descent over an objective function to add new models to the ensemble. This makes XGBoost more robust to overfitting and able to handle large datasets [3].

B. Model Hyper-parameters

All of the hyper-parameters were left on their default values except for the following:

- *eta*, which is used to prevent overfitting and is analogous to a learning rate,
- *max_depth*, which controls the maximum depth of each tree added to the ensemble, and
- *scale_pos_weight* which controls the balance of positive and negative weights.

IV. Methodology

A. Feature Selection

Three different sets of features were created for the purpose of training the models. The first set was all of the features (ALL), the second set was all the features that had high correlation (below -0.4 or above 0.4) with the class label (malware / benign) (CORR), and the third set of features were those features that had high mutual information (greater than 0.1) with the class label (M-INF). For simplicity in reporting the results, we shall refer to the models trained with the ALL, CORR, and M-INF feature sets as the ALL, CORR, and M-INF models, respectively.

B. Model Selection

For each set of features, a grid search on the parameters in Table I was performed; this means models with each possible combination of hyper-parameters from Table I were trained and tested using repeated stratified k-fold cross-validation with 2 repeats and 5 folds. The metric used for cross-validation was the f1 score, and the results of the best models for each feature are given in the next section. The hyper-parameters of the best models for each feature set are given in Table II. The value for scale_pos_weight of 1.71 in Table I was considered because this is the approximate ratio of benign to malware samples in the dataset.

TABLE I
Hyper-parameter grid

eta	0.1	0.3
max_depth	3	6
scale_pos_weight	1	1.71

TABLE II
Chosen hyper-parameters

feature set	eta	max_depth	scale_pos_weight
ALL	0.3	6	1.71
CORR	0.3	6	1
M-INF	0.1	6	1.71

C. Training and Testing

Before any model and feature selection process began, the dataset was split into two parts, 80% of the data was used for model selection and training the selected model, while the other 20% was used to test the model and estimate its performance. The data sampling was stratified according to the class label, meaning that the ratio of benign to malware samples stayed the same in both the training and the testing set.

V. Results

A. Metrics

For the purposes of testing the models, classification reports and confusion matrices were generated, also the precision-recall curves of the models were visualized. The classification reports contain the models’ precision, recall, and f1-score - because of the nature of malware detection, accuracy wasn’t considered since it just tells us how many of our predictions were correct (and since there is more benign software than malware, a model that doesn’t detect any malware could still have high accuracy). Precision and recall are much more important to consider since they tell us how much of the malware detected was really malware and how much of the present malware was detected, while the f1-score is calculated as a combination of precision and recall. From the confusion matrices, we can easily see if the model has a lot of false positives or false negatives while from the precision-recall curves, we can get a feel for the trade-off between the precision and recall of the model. In the lower-left corners of the precision- recall figures the average precision is also given which is the weighted average of the precision for each level of recall.

B. Classification Reports

From the classification report in Table III we can see that the ALL Model has both very high precision and very high recall for both malware and benign samples which means the model could be useful in practice since it neither misses a lot of the malware, nor does it misclassify a lot of the benign software.

TABLE III
Classification report for the ALL Model

	precision	recall	f1-score	support
benign	0.99	0.99	0.99	1896
malware	0.98	0.99	0.98	1111

The CORR Model seems close behind the ALL Model according to Table IV. It has lower precision and recall for malware samples, but nevertheless high enough that the model could be useful. Also since the model has fewer features it would require less time to process a given file.

TABLE IV
CLASSIFICATION REPORT FOR THE CORR MODEL

	precision	recall	f1-score	support
benign	0.94	0.96	0.95	1896
malware	0.93	0.90	0.91	1111

The M-INF Model has a low recall for the benign samples and very low precision for the malware samples as can be seen in Table V, and is overall the worst of the three models being compared.

TABLE V
CLASSIFICATION REPORT FOR THE M-INF MODEL

	precision	recall	f1-score	support
benign	0.95	0.75	0.84	1896
malware	0.68	0.93	0.79	1111

C. Confusion Matrices

From Table VI we can see that the ALL Model missed only 15 of the malware samples and misclassified only 20 of the benign samples, out of 1111 malware samples and 1896 benign samples which only reinforces the notion that the model is good and could be useful in practice - it has both little false negatives and false positives.

TABLE VI
CONFUSION MATRIX FOR THE ALL MODEL

actual / predicted	benign	malware
benign	1876	20
malware	15	1096

As we can see in Table VII compared to the other two models, the CORR Model missed most of the malware samples but didn't misclassify many of the benign samples, which is also important to consider. The fact that the model doesn't misclassify a lot of the benign samples, and only misses about 9.9% of the malware samples, only reinforces the notion that it might be useful in some scenarios in practice.

TABLE VII
CONFUSION MATRIX FOR THE CORR MODEL

actual / predicted	benign	malware
benign	1820	76
malware	110	1001

Table VIII shows us that even tho the M-INF Model misses less of the malware files than the CORR Model, it misclassifies almost one-quarter of the benign files making the model very unusable in practice since we don't want benign files being deleted for no reason (or at the very least scaring users).

TABLE VIII
CONFUSION MATRIX FOR THE M-INF MODEL

actual / predicted	benign	malware
benign	1421	475
malware	79	1032

D. Precision-Recall Curves

Figure 1 shows that the ALL Model has an excellent precision-recall curve - there aren't many trade-offs between the two values and the average precision of the model is 1 (lower left of Figure 1) which is quite outstanding.

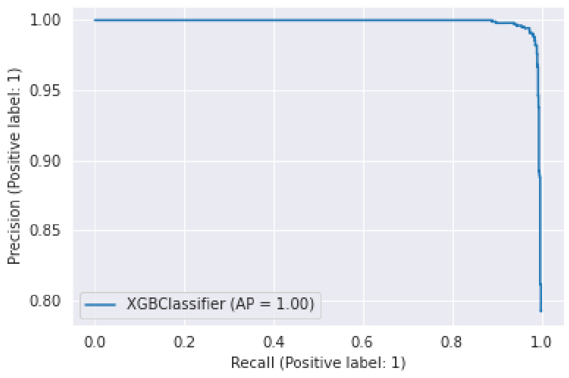

Fig. 1. Precision-recall curve of the ALL Model

The CORR Model has a bigger trade-off between precision and recall, as shown in Figure 2, but still good enough to be useful (the model has an average precision of 0.96) - this, combined with the fact that the model should require fewer computing resources, would mean the model could be useful on older hardware.

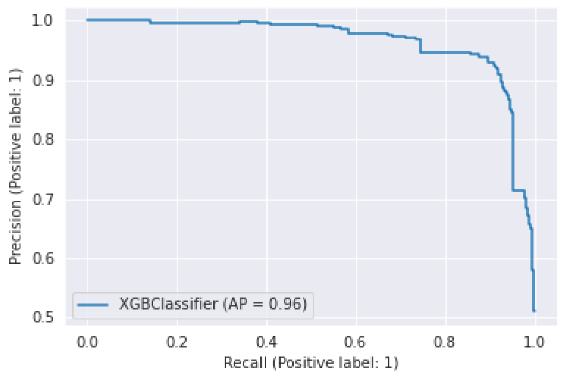

Fig. 2. Precision-recall curve of the CORR Model

From Figure 3, we can see that the M-INF Model has a high trade-off between precision and recall and the lowest average precision of all the models; this serves to reinforce the fact that this model would have almost no practical use.

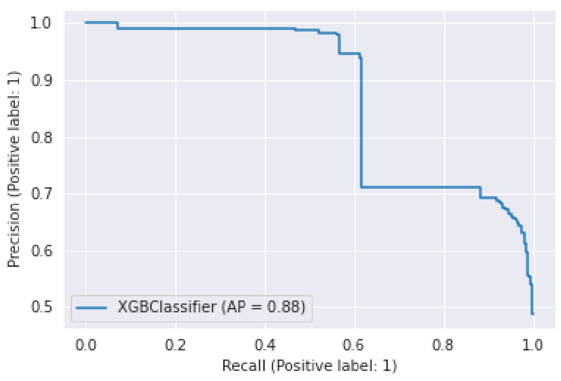

Fig. 3. Precision-recall curve of the M-INF Model

VI. Future Work

Possible next steps from this research include integrating the ALL Model or CORR Model in a mobile (or even desktop or web) application that can scan Android applications, trying to build an explainable model by using Shapley values to rank each feature's contribution (knowing why the model predicts something as benign or malware could be valuable information) which could also be integrated into a mobile (desktop or web) application, retraining and reevaluating the model on more recent data, and also trying to build an unsupervised model that could more easily be trained and retrained (since there is no need for manual labeling of the data). Another direction to go in form here would be adding more features (which would hopefully add more information and improve the models performance even further) and using dimensionality reduction techniques like PCA to reduce the vector space of the dataset with the goal of not sacrificing performance and training speed.

VII. Conclusion

Because of the fast-changing nature and growing complexity of malware, the amount of sensitive information kept on Android devices, and the inefficacy of conventional antivirus programs, other ways of securing Android devices are needed. The main advantage machine learning methods have over traditional detection of malware is that they can detect zero-day attacks since they aren't based on signatures (they need not have seen the malware before to be able to detect it). In this research, we have compared several different models for malware detection in Android applications and developed a model that can efficiently detect malware while also keeping the number of false positives relatively low - which is a very desirable quality of a malware detection system since we do not want to remove benign files needlessly. And although our model, or any machine learning model for that matter, cannot provide a 100% malware detection rate while keeping the false positives at 0%, we believe that machine learning might provide our best bet in defending against the ever-growing threat of rapidly evolving malware.

References

[1] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. Drebin: Effective and explainable detection of android malware in your pocket. In *Ndss*, volume 14, pages 23–26, 2014.

[2] Arpita Jadhav Bhatt and Neetu Sardana. Malicious ios apps detection through multi-criteria decision-making approach. 2022.

[3] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794. ACM, 2016.

[4] Jovana Dobreva, Aleksandra Popovska Mitrovikj, and Vesna Dimitrova. Maldewe: New malware website detector model based on natural language processing using balanced dataset. In *2021 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 766–770. IEEE, 2021.

[5] Ken Dunham. *Mobile malware attacks and defense*. Syngress, 2008.

[6] Qi Fang, Xiaohui Yang, and Ce Ji. A hybrid detection method for android malware. In *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, pages 2127–2132, 2019.

[7] Han Gao, Shaoyin Cheng, and Weiming Zhang. Gdroid: Android malware detection and classification with graph convolutional network. *Computers & Security*, 106:102264, 2021.

[8] Dragoş Gavriluţ, Mihai Cimpoeşu, Dan Anton, and Liviu Ciortuz. Malware detection using machine learning. In *2009 International Multiconference on Computer Science and Information Technology*, pages 735–741. IEEE, 2009.

[9] Manel Jerbi, Zaineb Chelly Dagdia, Slim Bechikh, and Lamjed Ben Said. Android malware detection as a bi-level problem. *Computers & Security*, 121:102825, 2022.

[10] Niall McLaughlin, Jesus Martinez del Rincon, BooJoong Kang, Suleiman Yerima, Paul Miller, Sakir Sezer, Yeganeh Safaei, Erik Trickel, Ziming Zhao, Adam Doupé, et al. Deep android malware detection. In *Proceedings of the seventh ACM on conference on data and application security and privacy*, pages 301–308, 2017.

[11] Justin Sahs and Latifur Khan. A machine learning approach to android malware detection. In *2012 European Intelligence and Security Informatics Conference*, pages 141–147. IEEE, 2012.

[12] Guillermo Suarez-Tangil, Juan E Tapiador, Pedro Peris-Lopez, and Arturo Ribagorda. Evolution, detection and analysis of malware for smart devices. *IEEE Communications Surveys & Tutorials*, 16(2):961–987, 2013.

[13] Fei Xiao, Yi Sun, Donggao Du, Xuelei Li, and Min Luo. A novel malware classification method based on crucial behavior. *Mathematical Problems in Engineering*, 2020, 2020.

[14] Suleiman Yerima. Android malware dataset for machine learning 2. *https://figshare.com/articles/dataset/Android_malware_dataset_for_machine_learning_2/5854653*, 2018.

[15] Suleiman Y Yerima and Sakir Sezer. Droidfusion: A novel multilevel classifier fusion approach for android malware detection. *IEEE transactions on cybernetics*, 49(2):453–466, 2018.