A novel dataset for fake android anti-malware detection

Saeed Seraj
Department of Computer Engineering
Yadegar-e-Imam Khomeini (RAH)
Shahre Rey Branch
Islamic Azad University
Tehran, Iran
S.Seraj@iausr.ac.ir

Michalis Pavlidis
School of Computing, Engineering and Mathematics
University of Brighton
Brighton, United Kingdom
M.Pavlidis@Brighton.ac.uk

Nikolaos Polatidis[†]
School of Computing, Engineering and Mathematics
University of Brighton
Brighton, United Kingdom
N.Polatidis@Brighton.ac.uk

ABSTRACT

Today in the world people are able to get all types of Android applications (apps) from the app store or various sources over the Internet. A large number of apps is being produced daily, some of which are infected with malware. Thus, the use of anti-malware identification tools is essential. At the same time, a number of attackers who exploit a number of anti-malwares have been doing obtaining information from mobile phones in various ways, such as decompiling or infecting anti-malware. Therefore, in this paper, we developed a classification dataset from collected anti-malware data looking for fraudulent anti-malware products. Additionally, we applied various machine learning algorithms and we propose a combination of algorithms which provides high accuracy over various evaluation tests, showing that our approach is both practical and effective.

CCS CONCEPTS

• **Management of Computing and Information Systems → Security and Protection**

KEYWORDS

Malware, Anti-malware, Fake anti-malware detection, Android, Cyber security, Machine learning

ACM Reference format:

1 Introduction

In this paper, we gathered information of existing claimed anti-malware android programs and examine if they have security issues. To this extend, we collected many android anti-malware programs from various sites such as Google Play and then classified them as a regular app or malware program using www.virustotal.com and almost 70 other reputed anti-malware detection engines. Hence, we developed a novel dataset including many anti-malware information derived from APK files including information about the permissions required by those apps. Our second contribution is an ensemble classifier which can detect and classify apps as normal program or malware. Moreover, for the analysis if data including android executable files, reverse engineering was performed on these files. Then, the required features were extracted and converted to the desired format suitable to analyze. In fact, the purpose was to analyze different batches of data and identify different subsets of malware in the Android operating system, which can help prevent intrusions into Android phones and prevent the hacking of the operating system or other files. There are two methods for detecting malware: static and dynamic. Static methods represent malware detection strategies based on an identification code. A common way in an antivirus program is to interrupt using a predetermined list of known attacks, but it is not able to detect all malicious cases because its database regularly needs to get updated. The main advantages of this method are accuracy and decrease the system overhead and runtime. One of the disadvantages of this method is that it cannot detect altered codes or malwares whose code has changed. In this method, the identification code is a hash code or a unique code that is stored in the database [11].. Dynamic methods analyze behavior or deviation. They check behavior of abnormal calls at the time of program execution or the granting of irrational access requests at the time of program installation. The advantage of this method is that it provides a useful understanding of how a malware produced and executed. In this method, suspicious objects are evaluated based on their activities in the system. In fact, irregular activities indicate that the target is suspicious or destructive. The purpose of good behavior is to encode or construct an object. Features of this method include detecting all kinds of malicious unknown attacks, storing complexity for behavior patterns, detecting flow dependency, time complexity detection, and polymorphic malware detection. This method works better than the identification code method, but it still has a lot of errors and in order to fix this problem, it is better to use machine learning techniques because machine learning techniques are able to detect new types of malwares. In addition, classification methods require numerous training examples to construct classification models [11]. In this paper we make the following contributions:

- A novel android fake anti-malware dataset is introduced.
- An ensemble classification method is applied to show the effectiveness of the data.

The rest of the paper is organized as follows: section 2 presents the related work, section 3 explains the dataset and describes the proposed classifier, section 4 contains the evaluation and section 5 is the conclusions.

2 Related work

In [1], a framework has been proposed that extracts logical file permissions, generates feature vectors, and uses six different rating tools to create distinct attributes. In this feature, the various categories of learning tools reduce the Data Mining tool, Weka used to classify Android apps. In [2], SIGPID, a malware detection system based on permission usage analysis, has been introduced to counter the rapid increase in the number of Android malware. Instead of extracting and analyzing all Android permissions, they used three levels of feature reduction by extracting permission data to identify more important permissions that can be used to distinguish between malicious and benign apps. Then, SIGPID is used by the classification method based on machine learning to classify various families of malicious and benign apps. In [3], a new method for extracting contrasting permission patterns was used to compare the difference between Android's malicious and benign apps based on permissions and use these differences to identify Android malware tools. Unlike existing systems, this is the first analysis of permissions that are required and used, then used the support-based candidate method to extract unique items, or from permission patterns for detection of Android malware. In this method, application patterns allow the identification of android malwares using a data set to cope with the problem of increasing the number of Android malware on various sources. In [4], the apps are categorized into three categories based on their APIs and permissions: safe, suspicious, and malicious, to achieve this, they set the three levels of analysis. Accordingly, the classification system based on API permissions performed based on YARA rule. API, class, and general methods of each of the extracted applications were from AndroidManifest.xml, Classes.dex, and compliance with the YARA rule. They ultimately provided user awareness by providing insight into application behaviors that allowed them to decide whether to install the application on their own devices. In [5], a new approach has been proposed that analyzes Android program features. This approach is performed by performing static analysis to extract features from an APK files. The extracted features are useful and meaningful to create efficient teaching systems. In addition, a permission-based model was introduced that uses a self-organizing map algorithm. In [6], the researchers focused on the many challenges that developers are posing when creating descriptions for permission usage. They have proposed a new framework, CLAP, that describes potential explanations of similar programs. CLAP provides information retrieval levers and text summarization techniques to find repeated permission usages. CLAP is a large dataset including 1.4 million Android apps. In [7], other known algorithms such as KNN and SVM were also evaluated in this study. However, since the J48 produces the highest level of accuracy, all experiments used this study. J48 is one of the top ten data mining algorithms that they also use to interpret the tree in order to see better features and to isolate software and malware. In training, the MalGenome dataset is used to represent malicious applications. In [8], to address problems regarding malware, they studied real-world Android apps to mine hidden patterns of malware and succeeded to extract highly sensitive APIs that are widely used in Android malware. In addition, they implemented an automated malware detection system, MalPat, to fight against malware and assist Android app marketplaces to address unknown malicious apps. In [9], they examined the effectiveness of exploring sandboxes in detecting malicious apps using five testing tools. They infected two sets of malicious and benign apps to check whether Sandbox based on sensitive APIs can effectively detect malicious behavior in their malicious apps. In [10], they described privacy issues by classifying application permissions and classification credentials using the Nave Bayes classification. Their results showed that they were categorized by GP-PP and validated through Classifiers Nave Bayes, users can decide which applications are safe to install, and which application requires what permission according to application generic (anti-malware). Permissions that fall into the Privacy Invasive class are accessing your personal information, camera, microphone, read information, Bluetooth, and your location. In [11], A systematic literary review of malware detection methods using data mining. The papers examined are categorized into two main categories; (1) identification code based (2) behavioral based approach. Malware detection approaches are compared and analyzed according to various factors such as classification methods, data analysis method, number of used data, accuracy factor and analysis. In paper [12], they propose DL-Droid, a deep learning system to detect malicious Android applications through dynamic analysis using stateful input generation. They performed experiments (benign and malware) on real devices. Moreover, experiments were also conducted to compare the detection performance and code coverage of the stateful input generation method with the commonly used stateless approach using the deep learning system. In paper [13], they propose a hybrid analysis measure called EspyDroid+1 that tackles the drawbacks of static analysis in analyzing the obfuscated and run-time dependent parameters of reflection APIs. EspyDroid+ incorporates Reflection Guide Static Slicing (RGSS), an efficient method to cope with exploration of large number of program paths by pruning non-relevant program paths and ensures that the resultant paths get executed during the subsequent dynamic analysis. In [14], they introduce MAMADROID, a static-analysis-based system that abstracts application's API calls to their class, package, or family, and builds a model from their sequences obtained from the call graph of an application as Markov chains. This makes sure that the model is more resilient to API changes and the features set is of manageable size. In paper [15], they suggest the use of a decompiled source code for malicious code classification. This decompiled source code provides deeper analysis opportunities and comprehension of malware nature. In [16], they propose a novel malware classification approach for malicious Android applications using RNNs and CNNs so that their model learns the generalized correlation between obfuscated string patterns from an application's package name and the name of the certificate owner. Their model extracts machine learning (ML) attributes using gated recurrent units (GRU), and an additional CNN unit further optimizes the attribute process of extraction. In [17], they proposed an innovative detection model, called PermPair, which compares constructs the graphs for malware and also normal samples by permission pairs extraction from an application manifest file.

The difference between this work and other related works is that in this paper as far as we are concerned, we are the first looking for fake anti-malware whereas the others are looking for malware and not fake anti-malware.

3 Dataset and proposed classifier description

3.1 Dataset

The dataset can be used to identify and anti-malwares through the use of appropriate classifiers. Therefore, it can be used in supervised learning environments. By fake anti-malware we mean software apps in Android that are in fact fake ones and they have been made for abusive purposes (e.g. steal information from a device or make it unresponsive) even though those fake ones act like a real one that users don't even notice or understand. Hence, the purpose of this research is to identify types of fake anti-malware and anti-malwares infected with malware. The dataset contains 1200 entries out of which 869 are negative and 331 are positive (i.e. pretending to be anti-malware but are malware). Moreover, the dataset contains 328 features with values 0 or 1 and the last (329th) value if the class which is either negative or positive and these entries are filled from various website such as Google Play and others. For example, features include internet access and various app permissions and the dataset is licensed under a CC-BY-NC 4.0 license and it is available online accompanied by a full explanation for everyone to understand[1].

3.2 Proposed classifier

For the classification of the final result we have used a combination of classifiers. More specifically, we have applied a Random Forest classifier with 100 estimators, and 2 neural network multi-layer perceptron classifiers with 1 hidden layer and 100 neurons in each and random weight initialization. The proposed ensemble classifier which works as follows (an arbitrary positive number of classifiers can be used, but in this case, we have used 3 classifiers):

1. Each classifier takes as input the training part of the dataset
2. Random forest classifier makes predictions
3. 1st MLP classifier makes predictions
4. 2nd MLP classifier makes predictions
5. The final predictions of each classifier are used to create a new dataset which is fed into a logistic regression classifier to make the final prediction

For the development of the algorithms the Python programming language has been used along with the Scikit learn machine learning library. The Random Forest classifier is an ensemble voting classifier and, in this case, it uses 10 decision trees and the majority of each decision tree votes gives the final result. For both MLP classifiers, we define an activation function as: $\boldsymbol{g}(\boldsymbol{z})$ with $\boldsymbol{x}$ input values and $\boldsymbol{w}$ weights as input. The activation function is shown in equation 1.

$$z = w_1x_1 + w_2x_2 + \cdots + w_mx_m \tag{1}$$

If g(z) is greater than a given threshold theta, the output is 1 or -1 otherwise, as shown in equation 2.

$$g(z) = \begin{cases} 1 \text{ if } z>\text{theta} \\ \text{-1 otherwise} \end{cases} \tag{2}$$

Where $z = w_1x_1 + w_2x_2 + \cdots + w_mx_m = \sum_{j=1}^{m} x_jw_j = w^Tx$

After that, Rosenblatt's perceptron rule is applied to update the weights as follows: Each of the weights is initialized with a small random number and for each iterative training step for each input x the output value is calculated, and the weights are updated. The output update is defined in equation 3, with $\Delta w_j = e\left(target(i) - output(i)\right) x_j^i$ and e is the learning rate, *target* the actual (true) class label and output the predicted label. Weight updates were iterative, and updates are performed at the same time.

$$w_j^+ = w_j + \Delta w_j \tag{3}$$

At the last step, the logistic regression linear model is used to make the final prediction.

4 Experimental evaluation

For the experimental evaluation we have used the Python programming language and the Scikit machine learning library. Moreover, we have used well known evaluation metrics such as the Accuracy, which is defined in equation 4, Precision which is defined in equation 5, Recall which is defined in equation 6 and F1 which is defined in equation 7. In equations 4, 5 and 6: TP is True Positive, TN is True Negative, FP is False Positive, and FN is False Negative. The results are presented in Table 1 for the accuracy, Table 2 for the Precision, Table 3 for the Recall and Table 4 for the F1. In all 4 tables We compare our proposed method with the Random Forest and MLP classifiers over 5 iterations using an 80/20 training testing approach and the average is presented at the last row. Overall our proposed method outperforms alternative classifiers in all metrics.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{4}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{5}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{6}$$

$$\text{F1} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{7}$$

No of execution	Random Forest	MLP Classifier	Proposed
1	95.4%	95%	95.8%

[1] https://www.kaggle.com/saeedseraj/antimalwarev1

2	94.5%	96.5%	96.6%
3	97%	95%	95.8%
4	94.5%	95%	98.3%
5	95%	95.4%	97.9%
Average	95.28%	95%	96.88%

Table 1: Accuracy results

No of execution	Random Forest	MLP Classifier	Proposed
1	95%	94.1%	95.3%
2	94.3%	95.3%	94.9%
3	96.9%	93.9%	95.4%
4	93.7%	94.5%	97.8%
5	93.3%	94.9%	96.9%
Average	95%	94.54%	96.06%

Table 2: Precision results

No of execution	Random Forest	MLP Classifier	Proposed
1	93.9%	94.6%	94.6%
2	92.3%	94.9%	94.9%
3	95.4%	93.9%	93.4%
4	92.4%	92.7%	97.8%
5	94.2%	93.7%	97.4%
Average	93.64%	93.96%	95.62%

Table 3: Recall results

No of execution	Random Forest	MLP Classifier	Proposed
1	94.4%	94.4%	94.9%
2	93.2%	95.1%	94.9%
3	96.1%	93.9%	94.3%
4	93%	93.6%	97.8%
5	93.7%	94.3%	97.1%
Average	94.08%	94.26%	95.8%

Table 4: F1 results

To further evaluate the quality of the proposed classifier we show in figures 1 and 2 a comparison of the proposed classifier against a variation using 1 MLP classifier instead of 2 and the logistic regression classifier at the end. The result averages for the accuracy and F1 metrics are as follows: Proposed method accuracy average over 5 iterations: **96.88%**. Proposed method with 1 MLP instead of 2 accuracy average over 5 iterations: **95.28%**. Proposed method F1 average over 5 iterations: **95.80%**. Proposed method with 1 MLP instead of 2 F1 average over 5 iterations: **93.80%**. This shows that using the proposed method with 2 MLP classifiers instead of 1 indeed makes a difference.

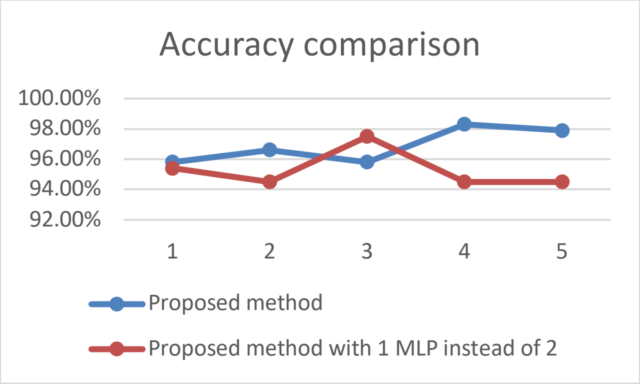

Figure 1: Accuracy results comparison between ensemble classifiers

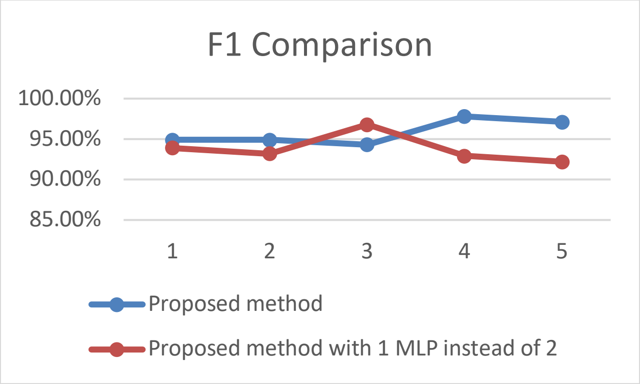

Figure 2: F1 results comparison between ensemble classifiers

5 Conclusions

In this paper we introduce a novel android fake-anti malware dataset. This dataset can be used to detect android fake-anti malware apps pretending to be legitimate but in reality, are malicious, through the use of appropriate classifiers. The results indicate that even with the application of pre-built classifiers such as Random Forest of MLP the results are very accurate. However, we propose the use of a combination of algorithms that further increase the accuracy and precision. Nowadays, the use of mobile devices and especially of those utilizing the Android operating system is very high and such a solution is necessary. However, such a methodology can be further extended, thus in the future we aim to look for more data that will allow us to classify by category and apply other novel classifiers.

REFERENCES

[1] Bhattacharya, A., & Goswami, R. T. (2017). Comparative analysis of different feature ranking techniques in data mining-based android malware detection. In Proceedings of the 5th International Conference on Frontiers in Intelligent Computing: Theory and Applications (pp. 39-49). Springer, Singapore.

[2] Li, J., Sun, L., Yan, Q., Li, Z., Srisa-an, W., & Ye, H. (2018). Significant permission identification for machine-learning-based android malware detection. IEEE Transactions on Industrial Informatics, 14(7), 3216-3225.

[3] Wang, C., Xu, Q., Lin, X., & Liu, S. (2018). Research on data mining of permissions mode for Android malware detection. Cluster Computing, 1-14.

[4] Park, J., Chun, H., & Jung, S. (2018, January). API and permission-based classification system for Android malware analysis. In 2018 International Conference on Information Networking (ICOIN) (pp. 930-935). IEEE.

[5] Sharma, A., & Doegar, A. (2018). Permission-Set Based Detection and Analysis of Android Malware. In Cyber Security: Proceedings of CSI 2015 (pp. 231-239). Springer Singapore.

[6] Liu, X., Leng, Y., Yang, W., Zhai, C., &Xie, T. (2018, August). Mining Android app descriptions for permission requirements recommendation. In 2018 IEEE 26th International Requirements Engineering Conference (RE) (pp. 147-158). IEEE.

[7] Sen, S., Aysan, A. I., & Clark, J. A. (2018). SAFEDroid: Using structural features for detecting Android malwares. In Security and Privacy in Communication Networks: SecureComm 2017 International Workshops, ATCS and SePrIoT, Niagara Falls, ON, Canada, October 22–25, 2017, Proceedings 13 (pp. 255-270). Springer International Publishing.

[8] Tao, G., Zheng, Z., Guo, Z., &Lyu, M. R. (2018). MalPat: Mining Patterns of Malicious and Benign Android Apps via Permission-Related APIs. IEEE Transactions on Reliability, 67(1), 355-369.

[9] Bao, L., Le, T. D. B., & Lo, D. (2018, March). Mining sandboxes: Are we there yet? In 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER) (pp. 445-455). IEEE.

[10] Kesswani, N., Lyu, H., & Zhang, Z. (2018). Analyzing Android App Privacy With GP-PP Model. IEEE Access, 6, 39541-39546.

[11] Souri, A., & Hosseini, R. (2018). A state-of-the-art survey of malware detection approaches using data mining techniques. Human-centric Computing and Information Sciences, 8(1), 3.

[12] Alzaylaee, M. K., Yerima, S. Y., &Sezer, S. (2020). DL-Droid: Deep learning based android malware detection using real devices. Computers & Security, 89, 101663.

[13] Gajrani, J., Agarwal, U., Laxmi, V., Bezawada, B., Gaur, M. S., Tripathi, M., & Zemmari, A. (2020). EspyDroid+: Precise reflection analysis of android apps. Computers & Security, 90, 101688.

[14] Onwuzurike, L., Mariconti, E., Andriotis, P., Cristofaro, E. D., Ross, G., & Stringhini, G. (2019). MaMaDroid: Detecting Android malware by building Markov chains of behavioral models (extended version). ACM Transactions on Privacy and Security (TOPS), 22(2), 1-34.

[15] Mateless, R., Rejabek, D., Margalit, O., & Moskovitch, R. (2020). Decompiled APK based malicious code classification. Future Generation Computer Systems.

[16] Lee, W. Y., Saxe, J., & Harang, R. (2019). SeqDroid: Obfuscated Android Malware Detection Using Stacked Convolutional and Recurrent Neural Networks. In Deep Learning Applications for Cyber Security (pp. 197-210). Springer, Cham.

[17] Arora, A., Peddoju, S. K., & Conti, M. (2019). PermPair: Android Malware Detection using Permission Pairs. IEEE Transactions on Information Forensics and Security.