University of Sindh Journal of Information and Communication Technology
(USJICT)
Volume 5, Issue 3, September 2021

ISSN-E: 2523-1235, ISSN-P: 2521-5582
Website: http://sujo.usindh.edu.pk/index.php/USJICT/

An Efficient Malware Detection Approach for Malicious Android Application

Madiha Amjad Hussain[1], Shariq Mahmood Khan[2], Shahzad Memon[3] And Syed Raza Hussain[4]
[1&2]*Department of Computer Science and Information Technology, NED University of Engineering and Technology, Karachi, Pakistan*
[3]*A.H.S. Bukhari Institute of Information & Communication Technology, Faculty of Engineering & Technology, University of Sindh, Jamshoro*
[4]*Department of Electronics Engineering, University of Sindh,Jamshoro, Pakistan*
E-mail: madhohussain@gmail.com[1]; shariq@neduet.edu.pk[2]; shahzad.memon@usindh.edu.pk[3] ; raza.shah@usindh.edu.pk[4]

Abstract: Artificial intelligence is changing the game for cybersecurity, analyzing enormous amount of risky data, increasing response times and enlarging the abilities of under-resourced security tasks. While security as IT percentage grows at a fast pace, the cost of security beaches grows at a more rapid pace. The malware targeting Android is growing. Android systems holds more than 70 percent of the market share.[1-3] This paper presents a simple APK analysis approach with the help of neural networks to identify malicious and benign application. The selected methodology is efficient in detecting malwares with an accuracy of 98.42% and false positive rate of 0.0121. malwares, Android, neural network, APK, security

Keywords: Malwares, Android, neural network, APK, security

I. Introduction

Androids due to its open specification not only facilitates application development and their release in the market share, but also gives rise to the threats and malwares on the Android platform.[4] The open specification of Android platform makes it quite impossible to administer Android operating systems and applications in a centralized way.[1, 5]

II. APK Analysis Approach for Dataset Preparation

Malwares in android are distributed through their terminals as APKs. Thus, analyzing APKs can be helpful in identifying android malwares. APK (android Application Package) provide preceding information important to identify malwares.[6, 7] Examining the permission requests and API calls in androidManifest.xml file is one method to recognize the android malware.[4, 8, 9] Another way is to use application cluster and description as the source[10,12,13]. Comprehensive study of different malware detection schemes are present in [14-18].

We have used analyzing APKs method to identify malware.In order to generate datasets for the analysis of Android applications APKs are required. These APKs can be obtained from various sources such as online APK providers, Google Play Store. There are a few alternatives which can be used for downloading the particular files, and there are certain techniques and Application Programming Interfaces exceptionally intended for this reason significantly considering.

Our main focus is on the *AndroidManifest.xml* and *classes.dex* files, as these two are the most suitable for the features required for the preparation of dataset. *AndroidManifest.xml* is used to extract permission request that are requested by application in order to run whereas *classes.dex* provides the data of API calls made in the application.[11] However, more data about the expected application can be gathered from *AndroidManifest.xml.*

For data to be reliable for the malware detection, the source from where APK is downloaded must be genuine and reliable. We have categorized our dataset into two classes: *benign(genuine)* and *malware* classes. For both (benign and malware) classes data is gathered from different sources. In order to collect benign APKs, one of the most reliable source is Google Playstore[1]. Apart from google play there are many other online sites that guarantee malicious free APKs – sites like APKMirror[2].

For the features, instead of taking descriptions of the APKs provided beneath the APK download option, we have used permission requests identified in the AndroidManifest.xml file and Application Performance Interface calls coded in the classes.dex file. In order to review the list of permissions required by the application we have opted to use Google official Integrated Environment of Google, Android Studio.

For the bytecode of API calls present in the *classes.dex*, we have used APKTool to convert bytecode into human-readable code, i.e., *smali* code. While the raw *classes.dex*

[1] https://play.google.com/store/apps

[2] https://www.apkmirror.com/

code itself can without much of a thought be considered as a series of Hex Numbers as is indicated in the documents of Android, this can not be considered as an optimal situation for training & classification.

Table 1 show the total attributes selected for the training of the model.

Label	Count
Permission Requests	136
API Calls	78
Total	214

Table 1– Total Number and Type of Features Selected

We have collected 214 features in order to develop dataset for our malware detection. Some of the most used API calls and Permission requests are shown in Table 2.

Feature	Percentage
READ_PHONE_STATE	95%
sendMultiPartTextMessage	80%
READ_EXTERNAL_STORAGE	98%
TelephonyManaer.getCallState	75%
INJECT_EVENTS	30%
CAMERA	97%
DELETE_PACKAGES	63%
WRITE_PROFILE	68%
Ljavax.crypto.spec.SecretKeySpec	51%
ACCESS_WIFI_STATE	89%

Table 2– Most Used API calls and Permission Requests in APKs in Gathered Data

III. Design of the Neural Network Scheme

This section explicitly demonstrate how neural networks are better in classification of Android application with the help of the vector representation. We compared the result of this approach to other AI algorithms using the very same dataset and evaluate the effectiveness of this approach with superior performance and flexibility.

A. Design

The model we have developed consist of 3 hidden layers with 500 nodes on each layer, an input layer and an output layer.
A binary neural network is defined as follows: From the training set samples

$D = \{x_n, y_n | x_n \in R^D, y_n \in \{0,1\}, n = 1 \ldots\ldots l\}$, a binary neural network learns a norm linear function

$$f(x) = \langle w, x \rangle + b \qquad (1)$$

determined by the threshold 'b' and weight vector 'w' by passing through the 3 hidden layers and activated by the activation function ReLU (Rectified Linear Unit), defined as:

$$f(x) = \max(0, x) \qquad (2)$$

While for the final output layer the activation function is described as:

$$f(x) = \frac{1}{1 + e^{-K}} \qquad (3)$$

for feed-forward.
The resultant of the output layer is compared with the expected output by the cost function

$$CE = -\sum_{i=1}^{c=2} t^2 \log x_i \qquad (4)$$

and is optimized to adjust weights and biases for minimizing the obtained square mean error for the resultant of the final layer for back propagation.

$$w_l = \beta_1 w_{l-1} + (1 - \beta_1) x_l \qquad (5)$$

$$b_l = \beta_2 b_{l-1} + (1 - \beta_2) x_l \qquad (6)$$

This feed-forward and back propagation constitutes one cycle (epoch) of neural network. The neural network consists of 100 cycles, each times a cycle is repeated, cost value is minimized.
After training phase, the neural network predicts the class (benign or malicious) of the testing sample (x) based on the decision function described below.

$$f(x) = \sum_{n=1}^{l} \alpha_n^* y_n K(x, x_n) + b \qquad (7)$$

If $f(x) > 0.5$, x is allocated to benign class, else it is allocated to malicious class.

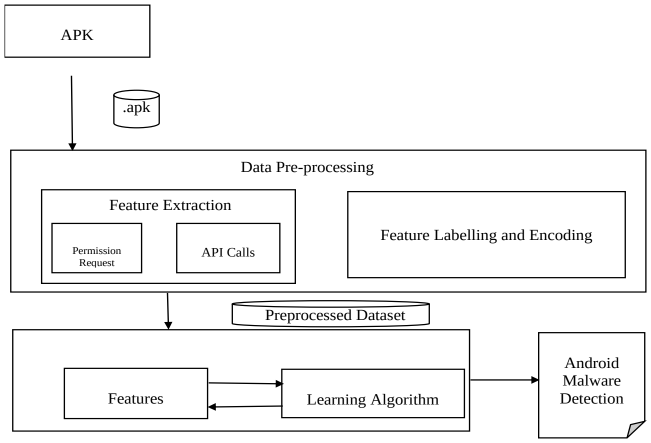

Figure 1: Architecural Design of the Proposed Model

B. Evaluation

To apprise the performance of the defined classifier, the data set used is haphazardly split into two sub sets: train set and test set and the examination is directed on it. In our examination 80% of the data is utilized for training while the staying 20% is utilized for testing reason, i.e., out of 3799 APKs dataset, the training set consist of 3039 APKs whereas the remaining 760 APKs are used for testing the classifier.

In the model defined Design, application data is passed through the input layer in the batches (500 input in each batch) for training. When the model is trained, test data is utilized to check the performance of the model

Four parameters are used to describe the effectiveness of the above described approach; True Positive (TP), False Positive (FP), True Negative (TN) and False Negative (FN). TP is the aggregate of correct benign APKs prediction, TN is the number of correct malicious APKs prediction, FP defines the aggregate of benign APKs forecasted as malicious and False Negative represents how many malicious APKs are incorrectly classified.

In Table 3, confusion matrix is utilized to portray the exhibition of the classifier utilized on test data to decide the true values for determining the performance of the schema.

		Actual Class	
		0	**1**
Predicted Class	**0**	487	5
	1	6	262

Table 3 – Confusion matrix

In Table 4, the overall generalization performance is shown. Thereby, describing the detailed accuracy by class

FP Rate	f1-score	Precision	Recall	Accuracy
0.0121	0.9831	0.9886	0.9776	98.42%

Table 4– Performance of Neural Network based Schema

IV. Performance Evaluation

This section verifies the accuracy and reliability of the proposed scheme through simulation and comparison of the performance with several well-known schemes. It also evaluates the performance of neural network with the help of different matrices and compares it with other known classification algorithms. It also address the possible issue which can affect the performance of the proposed model.

A. Model Comparison

In order for the baseline of the proposed model, the effectiveness of neural networks is compared with several classification standard algorithms. We have used Random Forest and Naïve Bayes classifiers as the comparative evaluator of the proposed model. Same dataset has been used in the above two mentioned models. The parameters used for evaluation; precision, recall and accuracy matrices are calculated for the comparison with the proposed model. Table 5 depicts the result of evaluation matrices of each model using the same dataset.

Model Name	Accuracy	Precision	Recall
Neural Networks	98.42%	98.86%	97.76%
Random Forest	92.24%	91.20%	90.38%
Naïve Bayes	69.21%	53.45%	98.13%

Table 5 - Performance Analysis of Proposed Model

According to the comparison with the other models, the proposed model effectiveness is much better because of its adaptive nature of learning. However, there are certain issues which must be addressed.

B. Precision Recall Analysis

Precision-Recall (PR) is a helpful measure for calculating successful prediction since the classes are imbalanced, i.e., we have greater amount of benign APKs data as compared to the malicious APKs data. As shown in Fig. 2, The high region under the curve determine both high precision and high recall, where high precision pinpoints low False Positive Rate (FPR), and high recall pinpoints a low False Negative Rate (FNR) determining that the proposed classifier evaluated has low chances of making wrong prediction concerning the security of Android through APKs.

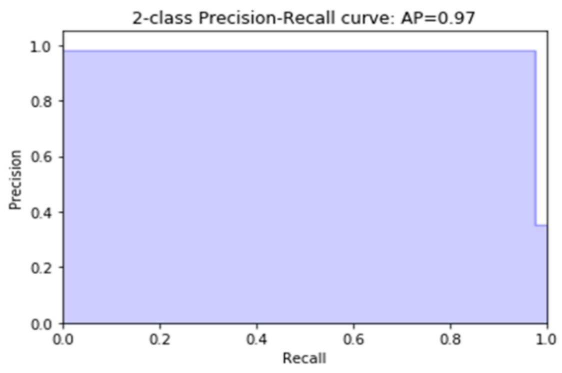

Figure 2 Precision-Recall (PR) Curve

C. Measure of Separability of Data

For the evaluation of performance measurement of the proposed classifier at various thresholds we have used True Positive Rate (TPR) & False Positive Rate (FPR) . Fig 3 shows the degree of the separability of the proposed classifier. It indicates the capability of classifier to distinguish between benign and malicious APKs.

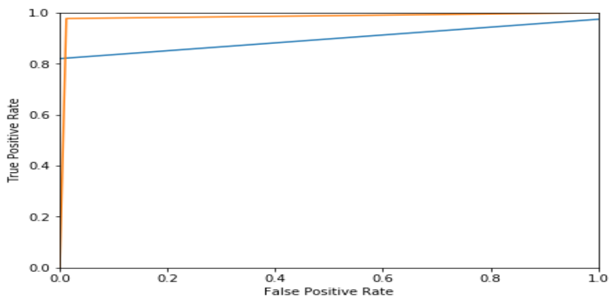

Figure 3 Degree of Separability

D. Issues and Limitations

Securing Android application through APKs analysis is an effective approach to promote security however there are certain limitations or issues of using machine learning approach. Following limitations and issues must be considered while using these approaches.

- False Positive (FP) and False Negative (FN) – False positive (FP) and false negative (FN) are inevitable in any machine learning approach despite of their excellent accuracy and precision. This can create a major risk for real world application. In order to reduce the risk, some additional measures are required.
- Dataset – Android application analysis requires quite large and quality dataset which is reliable and not manipulated.
- Labelling of Data – Using artificial intelligence, labelling is a must required. In this research, applications are labelled as benign or malware based on the permission requests and API calls of the application. Correct labelling is also essential for the effectiveness of the classifier.

V. Conclusion

In this paper, an efficient neural network scheme was presented for the identification of android malwares. The applicability of artificial intelligence for this purpose is described with the focus on neural network technique. As evident in the test performed on the proposed model, it is verified that the proposed model is able to learn features which are required to detect malwares with 98.42% accuracy having 98.86% precision, 97.76% recall and a very low false positive rate of 0.0121. Besides, it also outperforms other performed techniques to detect malwares in android applications. This study focuses on neural networks but other methods can also be evaluated for the identification of Android malwares.

References

[1] T. Takahashi and T. Ban, "Android Application Analysis Using Machine Learning Techniques," in *AI in Cybersecurity*. vol. 151, ed: Springer, pp. 181-205.

[2] S. Y. Yerima and S. Khan, "Longitudinal Performance Analysis of Machine Learning based Android Malware Detectors " presented at the International Conference on Cyber Security and Protection of Digital Services Oxford, United Kingdom, 2019.

[3] S. Alam, S. Yildirim, M. Hassan, and I. Sogukpinar, "Mininng Dominance Tree of API Calls for Detecting Android Malware," presented at the 2018 2nd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT), 2018.

[4] C. Chen, Y. Liu, B. Shen, and J.-J. Cheng, "Android Malware Detection Based on Static Behavior Feature Analysis," *Journal of Computers,* vol. 29, pp. 243-253, 2018.

[5] S. C. a. H. Yang, J. Jiang, Z. Ming, Z. Liang, and Z. Shan, "Research on Dynamic Safe Loading Techniques in Android Application Protection System," presented at the International Conference on Smart Computing and Communication, 2017.

[6] A. Ignatov, R. Timofte, W. Chou, K. Wang, M. Wu, Tim*, et al.*, "AI Benchmark: Running Deep Neural Networks on Android Smartphones," *ECV*, 2018.

[7] X. Xiao, S. Zhang, F. Mercaldo, G. Hu, and A. K. Sangaiah, "Android Malware Detection based on System Call Sequences and LSTM," *Multimedia Tools and Applications*, vol. 78, pp. 3979–3999, 2019.

[8] J. Jung, K. Lim, B. Kim, S.-j. Cho, S. Han, and K. Suh, "Detecting Malicious Android Apps using the Popularity and Relations of APIs," *IEEE Second International Conference,* 3-5 June 2019 2019.

[9] A. Pektaş and T. Acarman, "Deep learning for effective Android malware detection using API call graph embeddings," *Soft Computing*, pp. 1-17, 2019.

[10] R. A. Nix, "Applying Deep Learning Techniques to the Analysis of Android APKs," Master of Computer Science, The Department of Computer Science, Louisiana State University and Agricultural and Mechanical College, 2016.

[11] Huaxiaorong. (2019). *Android Application Package*. Available: https://en.wikipedia.org/wiki/Android_application_package

[12 Xue, D.; Li, J.; Wu, W.; Tian, Q.; Wang, J. Homology analysis of malware based on ensemble learning and multifeatures. PLoS ONE, 2019, 14, e0211373.

[13] Onwuzurike, L.; Mariconti, E.; Andriotis, P.; Cristofaro, E.D.; Ross, G.; Stringhini, G. Mamadroid: Detecting android malware by building markov chains of behavioral models (extended version). ACM Trans. Inf. Syst. Secur. 2019, 22, 1–34

[14] Z. Wang, Q. Liu, and Y. Chi, "Review of android malware detection based on deep learning," IEEE Access, vol. 8, pp. 181102–181126, 2020

[15] K. Liu, S. Xu, G. Xu, M. Zhang, D. Sun, and H. Liu, "A review of android malware detection approaches based on machine learning," IEEE Access, vol. 8, pp. 124579–124607, 2020

[16] E. C. Bayazit, O. K. Sahingoz, and B. Dogan, "Malware detection in Android systems with traditional machine learning models: a survey," in Proceedings of the 2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA), IEEE, Ankara, Turkey, June 2020.

[17] Chen, H.; Li, Z.; Jiang, Q.; Rasool, A.; Chen, L. A Hierarchical Approach for Android Malware Detection Using Authorization-Sensitive Features. Electronics 2021, 10, 432.

[18] Qing Wu, Xueling Zhu, Bo Liu, "A Survey of Android Malware Static Detection Technology Based on Machine Learning", Mobile Information Systems, vol. 2021, Article ID 8896013, 18 pages, 2021.