Dicle University Journal of Engineering

Volume 12 | Issue 2 Article 6

2021

AFWDroid: Deep Feature Extraction and Weighting for Android Malware Detection

Recep Sinan Arslan
Yozgat Bozok University, sinanarslanemail@gmail.com

Emre Ölmez
Yozgat Bozok University, emre.olmez@bozok.edu.tr

Orhan Er
Izmir Bakırcay University, orhan.er@bakircay.edu.tr

Recommended Citation

Arslan, Recep Sinan; Ölmez, Emre; and Er, Orhan (2021) "AFWDroid: Deep Feature Extraction and Weighting for Android Malware Detection," *Dicle University Journal of Engineering*: Vol. 12 : Iss. 2 , Article 6.
DOI: 10.24012/dumf.875036
Available at: https://duje.dicle.edu.tr/journal/vol12/iss2/6

Dicle University
Journal of Engineering
https://dergipark.org.tr/tr/pub/dumf
duje.dicle.edu.tr

Araştırma Makalesi / Research Article

AFWDroid: Deep Feature Extraction and Weighting for Android Malware Detection

Recep Sinan ARSLAN[1*], Emre Ölmez[2], Orhan ER[3]

[1] Yozgat Bozok University, Engineering Faculty,Department of Computer Engineering, Yozgat, Turkey, sinanarslanemail@gmail.com, ORCID 0000-0002-3028-0416
[2] Yozgat Bozok University, Yozgat, Turkey, emre.olmez@bozok.edu.tr, ORCID 0000-0003-1686-0251
[3] İzmir Bakırçay University, Engineering Faculty, Department of Computer Engineering, İzmir, Turkey, orhan.er@bakircay.edu.tr, ORCID 0000-0002-4732-9490

Doi: 10.24012/dumf.875036

ARTICLE INFO

Article history:

Received: 5 February 2021
Revised: 24 February 2021
Accepted: 1 March 2021

Keywords:

Feature weighting
Malware detection
Machine learning
Artificial Neural Network

ABSTRACT

Android malware detection is a critical and important problem that must be solved for a widely used operating system. Conventional machine learning techniques first extract some features from applications, then create classifiers to distinguish between malicious and benign applications. Most of the studies available today ignore the weighting of the obtained features. To overcome this problem, this study proposes a new software detection method based on weighting the data in feature vectors to be used in classification. To this end, firstly, the manifest file was read from the Android application package. Different features such as activities, services, permissions were extracted from the file, and for classification, a selection was made among these features. The parameters obtained as a result of selection were optimized by the deep neural network model. Studies revealed that through feature selection and weighting, better performance values could be achieved and more competitive results could be obtained in weight-sensitive classification.

Introduction

Development of mobile internet technologies, the prevalence of mobile devices is gradually increasing, and Android is leading this increase [1]. This open-source operating system, managed by Google, is available in more than half of mobile phones worldwide. The Android operating system has become a very important milestone in smartphones and mobile devices. It is used to install new features, add innovative features, and improve user experiences in many systems such as smartphones, tablets, smart TVs, car entertainment systems [2].

In the case that Android users do not like the stock firmware installed on their smartphones by the manufacturer, they can install a custom ROM. Some examples of custom ROMs are Omni ROM, Lineage OS, and AospExtended. While other smartphone platforms have a certain operating system and a number of applications that can be used depending on the operating system, the number of applications developed for each ROM within the Android ecosystem is quite high. These alternative systems do not always offer secure infrastructures [2].

Recently published statistics show that although the number of applications in the Playstore decreases in certain periods, it is generally increasing rapidly [3]. Besides, another feature of the Android platform is that applications can be downloaded and installed not only from Google Play Store but also from third-party platforms.

* Corresponding author
✉ sinanarslanemail@gmail.com

Please cite this article in press as R. S. Arslan, E. Ölmez, O. Er, “AFWDroid: Deep Feature Extraction and Weighting for Android Malware Detection”, DUJE, vol. 12, Iss.2, pp. 237-245, March 2021.

While applications downloaded and installed from the local application store can provide a reasonable level of security, this may not be possible for non-store apps. Applications developed for mobile devices are thought to be for the Android operating system, which corresponds to approximately 99% of the total applications [3][4]. More than 10 million android malware applications were produced in 2019. And it shows that 190.000 malicious apps occur monthly. In addition, in the first quarter of 2020, it was determined that an average of 480.000 malware appeared per month, showing a significant increase [5].

This structure offered by the Android system on the hardware and software side as well as the huge user base mentioned above has caused the emergence of malicious application developers for this operating system, the development of applications to exploit end-users with little experience of usage and increase in efforts to obtain information illegally. This whets the appetite of cybercriminals.

Detection of malicious applications developed for the Android operating system was easy in the early days of Android. By following the API calls of the application on a simple sandbox, non-complex malicious features of the application could be detected [6]. However, with Android 11, malware is engaged in increasingly more unpredictable activities and prefers more aggressive and complex techniques. On the other hand and in response to this, developers of malware detection systems are suggesting different detection techniques [7,8].

In the Android operating system, software developed with different ROMs and applications distributed from third parties are stored on end-user mobile devices. This requires the development of systems that use the most up-to-date methodologies to restrict or prevent access to sensitive personal information, to detect malware, and thus to secure mobile devices. In an environment where routine defense approaches fail to contain the ever-growing Android malware environment, these studies are critical.

Several basic approaches have been proposed in the literature for Android malware detection. These techniques are based on static, dynamic, and hybrid analysis.

Static analysis examines application codes, analyzes all possible execution paths, and aims to identify malicious codes before the application is run. It is not easy to obtain the codes of applications developed using modern compilers and runtime libraries. In addition, even if the codes are obtained, long analysis is required to make sense of the codes due to the obfuscation of codes. This has affected the performance of systems that detect malware by static analysis and caused an increase in false detection rates. In fact, static analysis can be bypassed by various obfuscation techniques, such as polymorphism, encryption, or packing. In addition, the applications analyzed by this technique are detected by comparing them with prebuilt signature databases. For this reason, these signature databases need to be kept up to date, and most importantly, it can be impossible to detect zero-day applications. Detection of real-time traces is possible only when the malware is executed [9].

Many studies in the literature have suggested the dynamic analysis approach to overcome these limitations of static analysis [10,11]. In this approach it analyses the behaviors of applications during their operations. This technique detects malware by analyzing the similarity between new and known behaviors of applications based on their application interface(API) calls. Polymorphic software, which is effective in changing static signatures, can be easily detected through active monitoring and detecting the behavior that cannot be hidden. This method, which has more useful features in terms of detection, creates a problematic situation for mobile devices with limited resources due to the excessive consumption of system resources and the emergence of a serious pre-processing phase during real-time monitoring.

Some studies have, therefore, suggested hybrid analysis for malware detection, which consumes fewer resources and makes a better classification. By combining the advantages of static and dynamic analysis, this method proposes a two-step approach [12].

Finally, Google created the Play Protect platform to limit malware invasion on Google Play Store. Thanks to this, it was possible to detect 700.000 malicious apps. More than 300 of these applications are used in Ddos attacks and are very dangerous. Good results have been achieved in the capture of 85 different adware families such as Stalkerware and were removed from Store. Play protect has a serious control over the PlayStore for malware detection. However, although it can detect many malignant applications, it has not been able to catch some types of them and has no activity on applications download from third party platforms [13].

Unlike previous studies in the literature, our contributions to Android malware detection can be summarized as follows:

• We propose a hybrid approach using static analysis and machine learning techniques for malware detection. Before the application was run, application features were obtained from the Manifest file, weighted, and selected in the pre-processing phase.

• A deep neural network was used to implement deep and broad feature learning. In this way, discriminative features were produced and classified based on the features obtained without examining the application code.

• The effect of feature weighting on classification performance and its contribution to the decrease in false positive(FP) value were shown.

• Repetitive and multi-group experiments were carried out with the basic data obtained, and the proposed method was compared with similar methods. Tests performed with 799 benign and 1081 malicious applications achieved a success rate of 99.6%.

In the rest of this study, current and similar studies for Android malware detection are mentioned. Thus, limitations with existing studies have been revelaed. Collection of dataset, extraction of application properties, pre-processing, ANN model and performance measurement metrics are explained in the methodology section. Then the tests and results were given in the light of the proposed model and a comparison was made with similar studies. In the conclusion, the study has been evaluated in general and suggestions for new studies were given.

Related Works

Many different security mechanisms are used on the Android platform. The most important of these is that app permissions must be granted by the user at install time. Thus, it is ensured that the user knows the permissions that the application will use. However, for consent-based privacy protection to be successful, end users must have sufficient awareness of security. This security infrastructure, which is excessively dependent on the user, creates protection problems. Therefore, antivirus software can also be used to ensure security. Thus, users are protected with signature-based protection. In an environment where malicious applications are increasing in quantity and variety rapidly, efforts are also made to develop more effective software detection systems.

In the study [14] conducted by Sasisharan, a behavioral-based approach for Android malware detection was proposed. The malicious dataset was compiled and coded to identify suspicious API classes. Patterns were created by performing multiple sequence aligments for different application families and applied to the hidden markov model(HMM) profile. 94.5% accuracy rate was achieved in the classification. A classical classification structure has been applied with new patterns.

MobiTive, is a real-time and sensitive malware detection tools using deep neural network[15]. It is pre-loaded on the mobile device and performs application scanning and monitoring. Since it is not possible to locate and operate traditional deep learning based approach on mobile devices, a different proposal has been made. Tests were carried out with LSTM, GRU and CNN networks, and as a result, the highest recognition rate was achieved in the GRU. Accuracy, precision and recall values were 96.75%, 96.78% and 96.72, respectively.

The study numbered [16] developed by Hossein et al. used the static analysis method. As a result of the analysis, feature vectors were created from information such as suspicious API calls, malicious activities, system calls, and purposes. The obtained features were classified using Gradient Boosting and deep learning methods. As a result, 97.3% of classification performance was achieved.

In Android applications, an attempt based on the principle of creating a permission-based security model was made to determine the permissions the applications request but do not use. While the requested permissions were obtained from the manifest file, the used permissions were extracted by code analysis. Thus, the extra requested permissions were revealed, and malware detection was performed. The study reported a 91.95% accurate classification [17].

DeepAMD [18] proposed an effective mechanism for detecting malware before it could be run, as static analysis requires. It adopted a method of classification with deep neural networks. It applied different approaches to detect and identify attacks that may occur at both static and dynamic analysis phases. A 93.4% accurate classification was achieved in the static layer, and 80.3% accurate classification was achieved in the dynamic layer. Also, in application category classification, an accuracy of 92.5% was obtained.

In malware detection, there is a lot of work on supervised or unsupervised feature learning, hierarchical feature extraction, and the application of deep learning to classify these features. Droidfusion [19] is an approach based on a multilevel architecture that enables the combination of base classifiers. Four different algorithms were proposed to classify the base classifiers, and then the results of these algorithms were combined and evaluated. Experiments were carried out with four separate datasets, and it was shown that more successful results were obtained than traditional models.

AndroidDialysis [20] proposed using the intents (implicit and explicit) of applications for malware detection. A dataset consisting of a total of 7405 applications, including 1846 benign and 5560 malicious applications, was used. Permissions were extracted from each application. Relationships between intents were evaluated along with permissions. Classification using permissions yielded an 83% success rate, but when evaluated together with intents, the success rate increased to 95.5%.

In the study conducted by Aloatibi, a multilevel malware detection method using regression coefficients was proposed to overcome some limitations of static and dynamic analysis approaches [21]. In the first layer, static analysis, and in the second layer, dynamic analysis was performed. Unlike other studies, a separate malicious application detection was performed for each layer. Machine learning techniques were used for classification. This proposed technique achieved 98.4%, 98.3%, and 99.0% success rates for accuracy, f-measure, and precision, respectively.

Methodology

This section is devoted to the description of the proposed model. The system has a multi-layered structure. The input layer, dex extraction, and preprocessing stages in this multi-layered structure were performed for three separate Android APK datasets. After this stage, in the preprocessing layer, a series of operations were carried out, such as extracting Dalvik EXecutable (DEX) files from the APK files, transforming them into bytecode format so that they could be used in the training phase, and generating the feature vectors. In the decision layer, the feature vectors obtained in the preprocessing stage were classified with a deep artificial neural network(ANN). Training, feature selection, detection, and evaluation steps are described in section 3.2. As a result of the operations performed in the model, the identification of the applications and the classification of the behaviors as benign or malicious were performed. The flow diagram of the model with a multi-layer architecture is shown in Figure 1.

Dataset Collection

Various methods have been developed for Android malware detection. Two of the malicious application sets used in these methods were selected and used in the training and testing processes of this study. The first of these families was the Drebin [22] dataset. It contains the permissions obtained from XML files as features. It was developed for use in dataset research and development and is distributed free of charge. It contains 5560 applications from 1000 different families. As the other malicious dataset, another widely used dataset, Genome [23], was used. Like Drebin, the Genome dataset is also distributed as open source. Finally, benign applications were downloaded from two popular application markets, Google Play [24] and Apkpure.com [25]. The most popular 900 applications were selected from various categories such as social media, news, finance, education, games, and sports. As shown in the dataset layer, three separate datasets were combined and evaluated in this study.

Feature Extraction

After the datasets were obtained, the manifest files were obtained from the DEX files of the applications. For the applications to be evaluated in the proposed model, 349 features were extracted for each application. The study numbered [26] shows that all features having the same weight negatively affect the classification performance. To overcome this, the features were weighted. Thus, it was prevented that all features have equal weight in the training phase and have the same effect at the model exit.

Preprocessing and Final Dataset Design

In the weighting phase, the frequencies of using separate features of 6820 malicious and 900 benign applications were taken into account. With these frequencies, it was aimed to pay more attention to the permissions frequently used by malicious and benign applications and to ensure that they have more effect in the model. As an example, it was aimed to ensure that the "FULL_INTERNET_ACCESS" permission and the "BATTERY_CHANGED" permission, which are widely used in malicious applications, do not have the same effect in understanding the intent of the application. The obtained usage frequencies were used for weighting the features. As a result, a 7720x350 two-dimensional feature vector containing the weighted feature data was obtained.

When this feature vector was examined, it was seen that some features had the same weight in all applications and that some applications had similar features. In this case, using the obtained vector for direct classification means that the same features and the same applications are repeatedly included in the training, which can cause the model to memorize. To overcome this problem, feature selection was performed, and applications with similar features were eliminated. As a result, a matrix containing 1081 malicious and 799 benign applications and 58 features was obtained. Of these applications, 70% were used for training, 15% for validation, and 15% for testing.

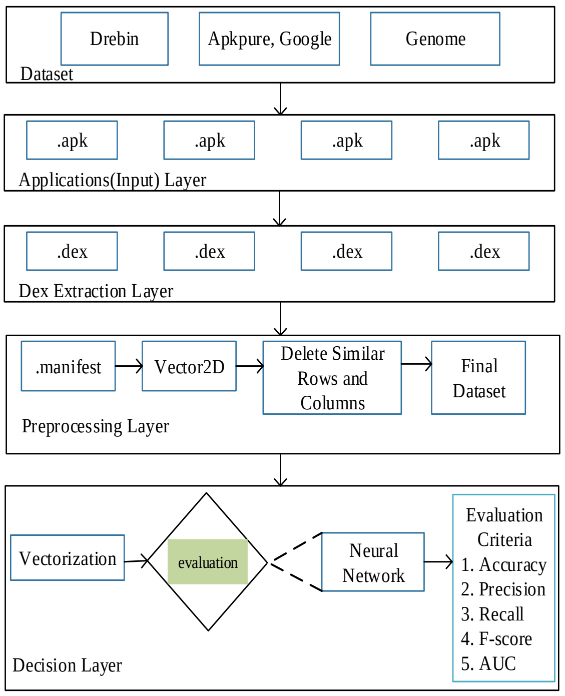


Figure 1. Layered diagram for Android Malware Detection

Decision Layer

To classify the weighted features obtained from the dataset used in this study, an ANN architecture was designed, as seen in Figure 2. The designed ANN consisted of two hidden layers with 10 neurons and an output layer with one neuron. The input layer of the ANN, on the other hand, had 58 features of each sample in the dataset, thus 58 neurons.

In the hidden layers of the designed ANN, the hyperbolic tangent sigmoid transfer function was used, and in the output layer, the logarithmic sigmoid transfer function was used. In the training phase, the Gradient Descent algorithm was used as the optimization algorithm with a learning rate of 0.01 and 1.000 epochs. The cross-entropy function was used as the cost function for performance evaluation during ANN training [27].To classify the weighted features obtained from the dataset used in this study, an ANN architecture was designed, as seen in Figure 2. The designed ANN consisted of two hidden layers with 10 neurons and an output layer with one neuron. The input layer of the ANN, on the other hand, had 58 features of each sample in the dataset, thus 58 neurons.

In the hidden layers of the designed ANN, the hyperbolic tangent sigmoid transfer function was used, and in the output layer, the logarithmic sigmoid transfer function was used. In the training phase, the Gradient Descent algorithm was used as the optimization algorithm with a learning rate of 0.01 and 1.000 epochs. The cross-entropy function was used as the cost function for performance evaluation during ANN training [27].

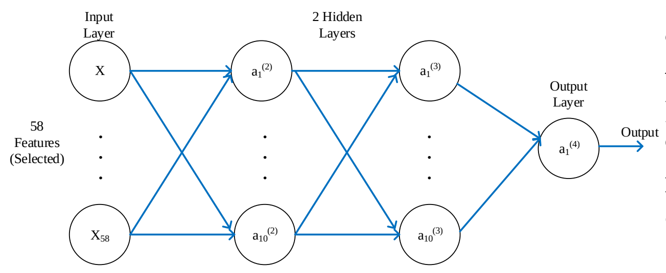


Figure 2. Architecture of Proposed Neural Network Model

Performance Calculation

To evaluate the efficiency of the proposed model, sensitivity, precision, accuracy, f-measure criteria were used. Accordingly, the equations given below represent definitions.

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

$$Sensitivity, Recall = \frac{TP}{TP + FN}$$

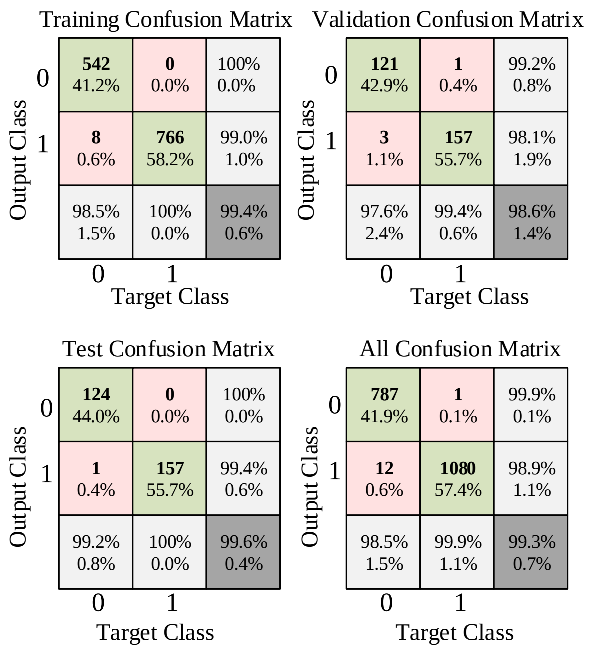
$$Precision = \frac{TP}{FP + TP}$$

$$F - measure = \frac{2 * TP}{2 * \ TP + FP + FN}$$

The TP value indicates how many of the positive test samples were correctly predicted positively and the FP indicates how many of the positive test samples were negatively prediced incorrectly. The TN value indicates how many of the negative test samplesa were correctly predicted negatively and The FN value indicated how many of the negative test samplesa were positively predicted incorrectly. In addition, the ROC curve is often used to evaluate the pros and cons of a classifier together. Expresses a graphical plot of specifity and sensitivity values. F-score is the measure of a model's accuracy in a dataset. It is used to evaluate systems that make binary classification of samples such as positive and negative.

Experimental Results

After the 1180 data in the dataset were randomly partitioned as 70% for training, 15% for testing, and 15% for validation, the performance of the ANN was analyzed. After the training, the success rate was 99.4% on the training set, 98.6% on the validation set, and 99.6% on the testing set, as seen in the confusion matrix in Figure 3. The success rate was found to be 99.3% on the entire dataset. As seen in these matrices, the results point to a high success rate classification, which indicates that the system is suitable for successful malware detection.



Training Confusion Matrix

Output Class \ Target Class	0	1	
0	**542** 41.2%	**0** 0.0%	100% 0.0%
1	**8** 0.6%	**766** 58.2%	99.0% 1.0%
	98.5% 1.5%	100% 0.0%	99.4% 0.6%

Validation Confusion Matrix

Output Class \ Target Class	0	1	
0	**121** 42.9%	**1** 0.4%	99.2% 0.8%
1	**3** 1.1%	**157** 55.7%	98.1% 1.9%
	97.6% 2.4%	99.4% 0.6%	98.6% 1.4%

Test Confusion Matrix

Output Class \ Target Class	0	1	
0	**124** 44.0%	**0** 0.0%	100% 0.0%
1	**1** 0.4%	**157** 55.7%	99.4% 0.6%
	99.2% 0.8%	100% 0.0%	99.6% 0.4%

All Confusion Matrix

Output Class \ Target Class	0	1	
0	**787** 41.9%	**1** 0.1%	99.9% 0.1%
1	**12** 0.6%	**1080** 57.4%	98.9% 1.1%
	98.5% 1.5%	99.9% 1.1%	99.3% 0.7%

Figure 3. Confusion Matrix

Table 1 demonstrates the calculated sensitivity, specificity, precision, and F1-score values for each subset of the dataset and the entire dataset. Accordingly, the best result belonged to the sensitivity value: A high rate of success was achieved with 1.0000 on the training set, 0.9937 on the validation set, 1.0000 on the testing set, and 0.9991 on the entire dataset.

Table 1. Metric Measurements on the Data Set

Metrics	Training Set	Validation Set	Test Set	Overall
Sensitivity	1.0000	0.9937	1.0000	0.9991
Specificity	0.9855	0.9758	0.9920	0.9850
Precision	0.9897	0.9813	0.9937	0.9890
F1 Score	0.9948	0.9874	0.9968	0.9940

Figure 4 shows the change in the cross-entropy cost function during training according to the number of iterations. Accordingly, the best validation performance was 0.066995 at epoch 1000.

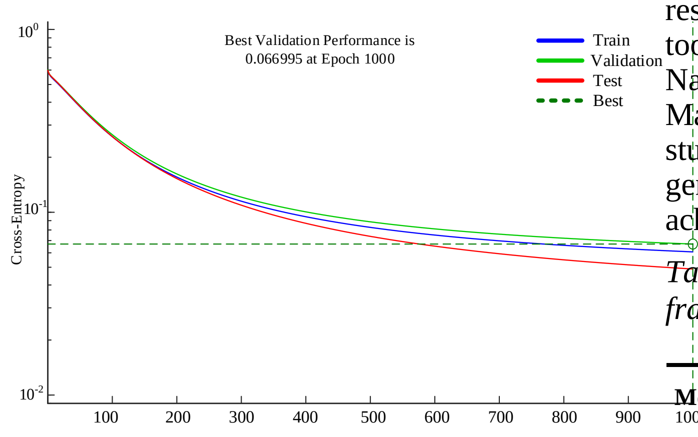


Figure 4. Cost function performance curve

Figure 5 shows the receiver operating characteristic curves. As can be inferred from these curves, the performances of the areas under training, validation, and testing curves (ROC) were very close to each other.

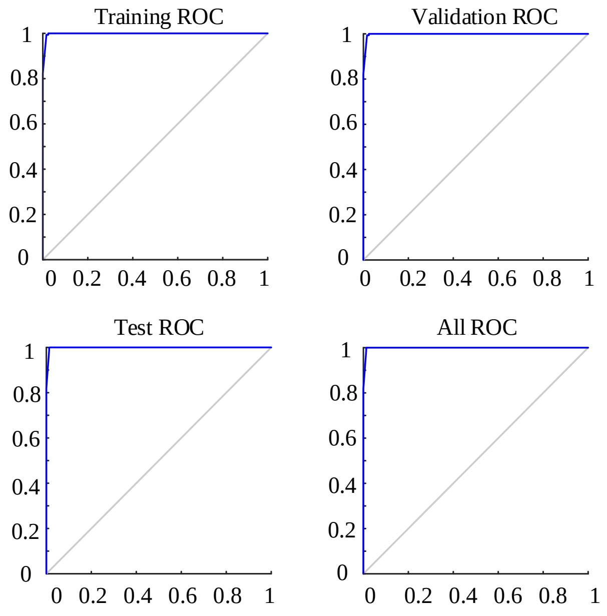


Figure 5. Receiver operating characteristic

Comparison with other proposed frameworks

We suggested the ANN model for Android malware detection. For this, the applications were analyzed and different tests were performed to try to obtain the best model. Experimental results of this model were obtained.

The comparison of the proposed model in this work with studies using different and up-to-date approaches is summarized in Table-2. When the results of known android malware detection tools such as Drebin[22], RevealDroid[28], Nauman [29], ProDroid[14], DL-Droid[30], Maldozer[31] were examned, although some studies obtained low classification rates, in general an accuracy of 95% and above has been achieved.

Table 2. AFWDroid vs other proposed frameworks

Metrics	[14]	[22]	[28]	[29]	[30]	[31]	AFWDroid
Accuracy	0.945	0.93	0.858	0.9	0.985	-	0.993
Precision	0.93	-	0.892	0.9	0.980	0.962	0.989
F1 Score	0.939	-	0.874	0.9	0.988	0.962	0.994

Conclusion and Future Works

This study proposed a novel method encompassing feature selection and feature weighting for malware detection in mobile applications. A multi-layer structure was used to apply the proposed model. Basically, static analysis-based processes were carried out to obtain application features. In this approach, firstly, malicious and benign application sets were obtained, and their features were extracted. Then, the obtained features were weighted separately for malicious and benign applications. Thus, it was aimed to prevent all features to be used in model training from having the same effect at the model exit. From the feature matrix obtained, the data that would cause problems during the training phase was eliminated, and thus, a more meaningful data feature set was obtained. Successful classification performance was then achieved with a weighted input cleared of noisy data. Various tests and conventional performance measurement methods were used to evaluate the success level of the proposed model. Thanks to its distinctive features, the proposed model both achieved a high level of success and enabled obtaining results rapidly.

Future studies are planned to try to take into account more features in feature weighting to further increase the classification performance, taking into account the level of correlation between the acquired features.

Acknowledgements

We would like to thank Drebin [22] and Genome [23] projects for providing malicious datasets free of charge and for their valuable contributions to the conduct of the study.

References

[1] S. Wang, Z. Chen, Q. Yan, K. Ji, L. Peng, B. Yang and M. Conti, “Deep and broad URL feature mining for android malware detection”, Information Sciences, 513, 600-613, 2020.

[2] M. Amin, T. A. Tanveer, M. Tehseen, M. Khan, F. A. Khan and S. Anwar, “Static malware detection and attribution in android byte-code through and end to end deep system”, Future generation computer systems, 102, 112-126, 2020.

[3] J. Clement, “statista.com,” [Online]. Available:https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/.

[4] F-Secure Team, “f-secure.com” [Online]. Available: https://blog.f-secure.com/another-reason-99-percent-of-mobile-malware-targets-androids/.

[5] J. Johnson, “statista.com,” [Online]. Available: https://www.statista.com/statistics/680705/global-android-malware-volume/

[6] R.S. Arslan, İ. A. Doğru and N. Barışçı, “Permission comparison based malware detection system for Android mobile applications”, Journal of Polytechnic, 20(1), 175-189, 2017.

[7] A.T. Kabakuş and İ.A. Doğru, “An in-depth analysis of Android malware using hybrid techniques”, Digital Investigation, 24, 25-33, 2018.

[8] İ. A. Doğru and Ö. Kiraz, “Web-based android malicious software detection and classification system”, Applied Sciences, 8(9), 1622- 1641, 2018.

[9] M. Jerbi and Z. C. Dagdia, “On the use of artificial malicious patterns for android malware detection”, Computer & Security, 92, 1-22, 2020.

[10] C. Willems, T. Holz and F. Freiling, “Toward autmated dynamic malware analysis using cwsandbox”, IEEE Security and Privacy Magazine, 5(2), 32-39, 2007.

[11] K. Rieck, T. Holz, C. Willems and P. Düssel, “Learning and classification of malware behaviour”, Proceedings of the 5th International Conference on Detection of Instrusions and Malware, and Vulnerability Assessment, 1-20, 2008.

[12] M. Wozniak, M. Grana and Emilio Corchado, “A survey of multip classifier systems as hybrid systems”, Information Fusion, 16(1), 3-17, 2014.

[13] A. Mathur, L.M. Podila, K. Kurkarni, Q. Niyaz, A.Y. Javaid, “NATICUSdroid: A malware detection framework for Android using native and custom permissions”, Journal of Information Security and Applications, 58, 1-14, 2021.

[14] S. K. Sasidharan, C. Thomas, “ProDroid – An Android malware detection framework based on profile hidden markov model”, Pervasive and Mobile Computing, 72, 1-16, 2021.

[15] R. Feng, S. Chen, X. Xie, G. Meng, S.W. Lin ve Y. Liu, “A performance-sensitive malware detection system using deep learning on mobile devices”, Information forensics and security, 16, 1-16, 2021.

[16] F. Hossein, C. Mauro, Y. Danfeng and S. Alessandro, “Anastasion: android malware detection using static analysis of appication”, 8th IFIP International Conference on New Technologies, Mobility and Security (NTMS), 1-5, 2016.

[17] R.S. Arslan, İ. A. Doğru and N. Barışçı, "Permission-based malware detection system for android using machine learning techniques", International journal of software engineering and knowledge engineering, 29(01), 43-61, 2019.

[18] S. I. Imtiaz, S. Rehman, A. R. Javed, Z. Jalil, X. Liu and W. S. Alnumay, "DeepAMD: Detection and Identification of Android Malware using high-efficient Deep Artificial Neural Network", Future Generation computer systems, 115, 844-856, 2020.

[19] S. Y. Yerima and S. Sezer, "DroidFusion: A Novel Multilevel Classifier Fusion Approach for Android Malware Detection", IEEE Transactions on Cynernetics, 49(2), 453-466, 2019.

[20] A. Feizollah, N. B. Anuar, R. Salleh, G. S. Tangil and S. Furnel, "AndroDialysis: Analysis of Android Intent Effectiveness in Malware Detection", Computers & Security, 65, 121-134, 2017.

[21] S. S. Alotaibi, "Regression coefficients as triad scale for malware detection", Computers and Electrical Engineering, 1-14, 2020.

[22] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, K.Rieck and C. Siemens, "Drebin: Effective and explainable detection of android malware in your pocket", Proceedings of the Annual Symposium on Network and Distributed System Security, 2014.

[23] Malgenome Project, "malgenomeproject.org,"[Online]. Available: http://www.malgenomeproject.org/.

[24] Google, "Google play store," [Online]. Available: https://play.google.com/store/apps?hl=en.

[25] APKPure Team, "APKPure.com," [Online]. Available: https://apkpure.com/cn/.

[26] L. Cai, Y. Li and Z. Xiong, "JOWMDroid: Android malware detection based on feature weighting with joint optimization of weight-mapping and classifier parameters", Computer & Security, 100, 1-14, 2020.

[27] E. Ölmez, V. Akdoğan, M. Korkmaz and O. Er, "Automatic Segmentation of Meniscus in Multispectral MRI Using Regions with Convolutional Neural Network (R-CNN)", Journal of Digital Imaging, 33, 916-929, 2020.

[28] J. Garcia, M. Hammad, B. Pedrood, A. Bagheri-Khaligh, S. Malek, "Obfuscation-resilient, efficient, and accurate detection and family identification of android malware", Technical Report, Department of Computer Science, George Mason University, 1-15, 2015.

[29] M. Nauman, T.A. Tanveer, Sohail. K, Toqeer. A., "Deep neural architectures for large scale android malware analysis", Cluster Computing Springer, 1-20, 2017,

[30] M. K. Alzaylaee, S. Yerima, S. Sezer, "Dl-droid: deep learning based android malware detection using real devices", Computer and Security, 89,1-11,2020.

[31] E.B. Karbab, M. Debbabi, A. Derhab, D. Mouheb, "Maldozer:automatic framework for android malware detection using deep learning", Digital investigation, 24, 48-59, 2018.