Advances in Parallel Computing Technologies and Applications
D.J. Hemanth et al. (Eds.)

doi:10.3233/APC210133

Deep Learning Based Static Analysis of Malwares in Android Applications

Nivedha K [a,1], Indra Gandhi K [a], Shibi S [a] Nithesh V [a] and Ashwin M [a]
[a] *Department of Information, Science and Technology, Anna University, Chennai, India*

Abstract. Android is a widely distributed mobile operating system developed especially for mobile devices with touch screens. It is an open source, Google-distributed Linux-based mobile operating system. Since Android is open source, it enables Android devices to be targeted effectively by malware developers. Third-party markets do not search for malicious applications in their databases, so installing Android Application Packages (APKs) from these uncontrolled market places is often risky. Without user's notice, these malware infected applications gain access to private user data, send text messages that costs the user, or hide malware apk file inside another application. The total number of new samples of Android malware amounted to 482,579 per month as of March 2020. In this paper deep learning approach that focuses on malware detection in android apps to protect data on user devices. We use different static features that are present in an Android application for the implementation of the proposed system. The system extracts various static features and gives them to the classifier for deep learning and shows the results. This proposed system will assist users in checking applications that are not downloaded from the official market.

Keywords. Android, APKs, Malware Detection, Third-party markets, Deep Learning.

1. Introduction

In smartphones and tablets, Android is the world's best-selling Operating System. The presence of number of markets present in Android like Google Play Store and other third-party markets also encourages the malware developers to develop a malicious application. The number of malware samples targeting Android has risen over the last few years. The reason is because there are a lot of third-party markets that are not monitored and regulated properly. The open architecture of Android enables users to install apps that do not inherently come from the Google Play Store. We can estimate that over 20,000 new apps are released every month, with over 1 million apps available for download from Google's official market, and probably another million distributed across third-party app stores. An attacker can pick up and change any benign application and upload its code to third-party markets, making users think it is not a malicious application. Android is known for its free, ready-made, low-cost and open source license, which was developed

[1] Nivedha K, Department of Information, Science and Technology, Anna University, Chennai.
E-mail: niveda0394@gmail.com.

by Google. Application support is the main drawback in Android, given its capabilities. It usually takes months before it reaches the user if a new version of the Android operating system or a security patch is issued [1]. They don't really reach out at all. This is due to the large range of Android device manufacturers and hardware. In the machine learning community, Deep Learning (DL) is gaining more and more attention. In the field of Image detection, Natural Language Processing and Speech Recognition, Deep Learning Classifiers have inspired a great number of successful approaches. In order to improve detection accuracy, DL classifiers have also been used recently for malware analysis. Since Deep Learning models are trained by feature learning rather than task-specific algorithms, they can recognize more characteristics than traditional machine learning techniques.

2. Related work

Static analysis detects malware based on the information that is present in an APK file. The static analysis features and techniques for detecting android malware is widely explored in this section.

Hota et al. [2] explained static analysis as the analysis of software without actually executing the program. Static analysis is one of the malware detection techniques that uses less computation. They read the bytes that are present in the dex file as input and they one hot encoded those bytes and then fed them to a Long Short Term Memory Network (LSTM). The final output is used to classify a particular application as malware or benign. They were able to achieve 95.3% accuracy. Also Naway et al. [3] provides a detailed explanation about the static features and their role in the detection of malware in android. The authors of [4,5] discussed permission misuse by android apps using a static analysis tool of identification stating that it is possible to obtain all the manifest file permission. In order to gain understanding of the purpose of the study, they review and give the reader the required preliminary information on android and static analysis. They explain the concept of static program analysis, permissions and analysis technique. The findings of this job can help encourage Android malware detection studies based on techniques of deep learning. Also [6,7] designed a more simplified instruction set since there are a lot of Dalvik opcodes to process. They analysed multiple smali files and discovered that there certain instructions have multiple opcodes because of parameters. For example, the same opcode will be used different operands, since operands maybe 8-bit or 16-bit. Based on the frequency of occurrence and core semantics they have identified 107 Dalvik opcodes. Then, group the similar instructions together using a single character. For example G for all the branch statements goto16, goto/32 since they do the same job. Then they use apk tool to get the smali files from the apk file. They then extract all the opcodes from the reduced instruction set and they map the opcodes to their corresponding characters.

3. Problem description

The architecture of android malware detection using static analysis is given in Figure 1. Two static analysis models are developed for detecting android malware.

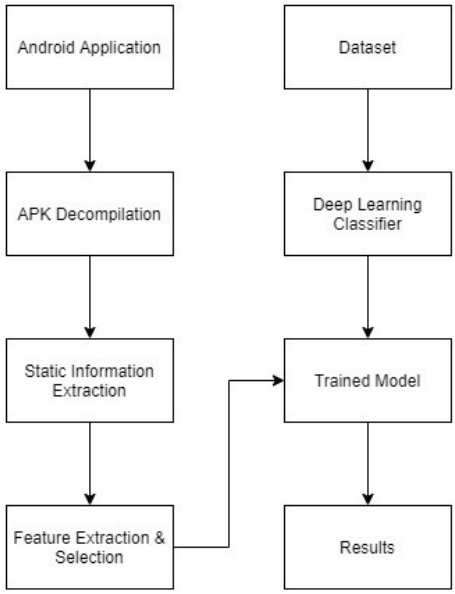

Figure 1. Android malware detection using static analysis

For training the first model, DREBIN dataset is used. It is a popular dataset used for static analysis of android malware detection. The dataset contains 5,560 applications from 179 different malware families. The samples were collected between August 2010 and October 2012, and the MobileSandbox project made them available to DREBIN. For training the second model, a 3gram opcode dataset is used for detecting android malwares.

Deep Neural Network algorithm is used to train both the models. The first one to detect android malwares using the static features Intents, API calls and permissions [8]. The second one to detect android malwares using the static feature opcodes [9,10]. The models are saved once it is trained with good accuracy. If an application has to classified as either benign or malware, the apk file of that particular application is decompiled and features are extracted from the application, then the saved model is loaded and provided with the feature set for prediction [11].

We decompile the APK file using tools such as Androguard and Apktool to extract static information about a specific android application. After decompilation, we will be able to get the manifest file, smali files (baksmali disassembler is used on dex file to obtain smali files) consisting of dalvik opcodes, the methods and classes used for the development of the application from the dex file.

After decompilation, we extract static information such as intents from the manifest file, API calls , permissions using the androguard tool and smali files for opcodes using apktool.

- Permissions: Applications use permissions in order to access the system resources.
- Intents: Intents are messaging objects that transfer data from one activity to another.
- APIs: APIs are the libraries and methods present inside the source code of an APK file.

- Opcodes: The Opcodes are instructions that an application executes.

We convert the information to features after static information is extracted. Once the features are extracted, by comparing the dataset, we select the important features from them. The Table 1 and 2 consist of feature set and number of features for building the first model and second model.

Table 1. Feature set for model building the first model

Feature Set	Number of Features
Permissions	113
Intents	23
API calls	71

Table 2. Feature set for model building the second model

Feature Set	Number of Features
3 gram opcodes	343

4. Methodology

4.1. First Model

Drebin dataset [12] was used for this model. It consists of 9476 benign applications and 5560 malware applications. It consists of 215 features. 5 malware apps had null values in some columns so those 5 apps were removed and 5555 malware applications. 6 commands feature was removed from the dataset because it was difficult to extract the commands using androguard. Some of the APIs in the dataset contains classname and method name so the classname is removed and the method name alone is taken as a feature. 2 API features HttpPost.init and HttpGet.init were removed because both of them have the same method name so if class name is removed there will be two features with the same name. Finally a total of 208 features was used for training the model.

Using the androguard tool, we decompile a given apk file and extract all the permissions present in the apk file using a method in androguard and take only the important permission keywords. Then we extract all the classes in the apk file, take only the relevant keywords, and API calls are classes in androguard that are marked as EXTERNAL, so we only take the classes marked as EXTERNAL. Next, using the Androguard tool, we get all the methods present in the APK file. We then read the contents of the manifest file using androguard and extract only the strings that are present within double quotes using regex. After that, using the intent keyword, we extract only the intents present from those values. Lastly, we select only a few of them from all the extracted features by comparing the dataset. Then we give the feature set to the deep learning model for prediction.

The model is trained using the Deep Neural Network algorithm (DNN). The model consists of an input layer with 207 neurons, one hidden layer with 100 neurons with ReLU activation function. As it is a binary classification, the output layer consists of one neuron and it uses the sigmoid activation function. The model is tested with 20 percent of the dataset split before the training of model.

Algorithm 1 Feature Extraction (First Model)

1: Decompile the given apk using the androguard tool.
2: Extract the permissions using androguard.
3: Remove ”android.permission.” string from all permissions and store it (Permissions).
4: Extract all the classes using androguard.
5: Split the classes using the separator and store only those classes that are marked as EXTERNAL (APIs)
6: Extract all the methods using androguard (APIs)
7: Get the contents of the manifest file using androguard
8: Using regex get only the strings that are present inside double quotes
9: Store all the strings that has ”intent.action” string in it (Intents)

4.2. Second Model

A 3gram opcode dataset downloaded from [13] was used for this model. It consists of data for 334 applications out of which 180 applications are malware and 154 applications are benign. All the 343 features are used for training the model.

Apktool is used for decompiling the apk file. After decompilation, we will get a classes.dex file so when we read a particular method in that dex file the opcodes will be in hexadecimal sequences, so in order to convert them into human readble form apktool uses a disassembler called baksmali which further decompiles classes.dex into several smali files. We read all the files inside the smali folder and store them word by word in an array. Then we read the opcode sequences that are present inside methods. A method starts with a ”.method” opcode and ends with ”end” opcode. Since each opcode have multiple versions we group them together and denote using a single character. Select only the 3gram opcode sequences that are present inside the methods.

Algorithm 2 Feature Extraction (Second Model)

1: Decompile the given apk using the androguard tool.
2: Read all the smali files line by line inside the smali folder.
3: Split the line based on blank space and store all the opcodes in an array
4: Read the opcode sequences present in between ”.method” and ”end” opcode.
5: **if** opcode is a move instruction **then**
6: denote them as ”M”,
7: goto instruction as ”G”,
8: **if** condition instruction as ”I”, getter instruction as ”T”, setter instruction as ”S” and method **then**
9: invoke instruction as ”V”, else ignore the opcode and read the next one.
10: Store the opcode sequence only if the length is 3.

The model is trained using the Deep Neural Network algorithm. The model consists of an input layer with 343 neurons, three hidden layers, one with 300 neurons and ReLU activation function and the other 2 hidden layers uses sigmoid activation function and consists of 150 and 100 neurons respectively. As it is a binary classification, the output layer consists of one neuron and it uses the sigmoid activation function. The model is tested with different benign apps obtained from Google Play Store and malware apps obtained from GitHub repository.

5. Results

In this paper, we implemented android malware detection using static analysis where we classify any android application to be malware or benign. Deep Neural Network has been used as the classifier in both the models The first model is trained with a training data set that contains approximately 12,000 applications and tested with 3000 apps. The first model was able to achieve an accuracy of 99.37 percent while tested with the 3000 applications. The second model is trained with a 3 gram opcode dataset that contains around 300 applications and tested with around 30 applications. The second model was able to achieve an accuracy of 97 percent. But since different combination of layers and neurons produced the same accuracy we evaluated the model using applications downloaded from Google Play Store and malware apps downloaded from GitHub repository.

5.1. Performance Analysis of First Model

This section discusses about the performance metrics of the proposed first android malware detection model and states about the confusion matrix, false positives(apps that are not malware but predicted as malware) and false negatives(apps that are not benign but predicted as benign) are explained.

A confusion matrix is one of the important classification metric that provides summary of prediction results. For each class, the number of predictions is given along with the correct and incorrect ones.

$$Confusion = \begin{bmatrix} 1921 & 9 \\ 10 & 1067 \end{bmatrix}$$

False Malware - Predicted Malware but Benign = 9
False Benign - Predicted Benign but Malware = 10
True Malware - Predicted Malware and it is true = 1067
True Benign – Predicted Benign and it is true = 1921

$$FalsePositiveRate(FPR) = \frac{FalseMalware}{FalseMalware+Truebenign} = 0.0047$$

$$FalseNegativeRate(FNR) = \frac{FalseBenign}{FalseBenign+TrueMalware} = 0.0093$$

$$\text{Accuracy}=\frac{TrueMalware+TrueBenign}{Totalno.Offsetsamples} = 0.9937$$

$$\text{Precision}=\frac{TrueMalware}{TrueMalware+FalseMalware} = 0.9937$$

$$\text{Recall}=\frac{TrueMalware}{TrueMalware+FalseBenign} = 0.9937$$

$$\text{F1 Score}=\frac{2*(Precision*Recall)}{Pecision+Recall} = 0.9912$$

By using the machine learning classifiers, we classify any android application to be benign or malware. With the help of confusion matrix we can predict false malware, true malware, false benign and true benign. According to this classification metrics like False Positive Rate (FPR), False Negative Rate (FNR), Accuracy, Precision, Recall and F1 Score has been calculated and compared with the proposed and existing machine learning classifiers. The proposed DL classifiers has better results when compared with existing classifies and is shown in Table 3.

Table 3. Comparison of DL classifier Vs other machine learning classifiers

ML classifiers / Metrics	FPR	FNR	Accuracy	Precision	Recall	F1
Logistic Regression	0.0109	0.0418	0.9781	0.9781	0.9582	0.9690
Decision Tree	0.0238	0.0251	0.9757	0.9757	0.9749	0.9664
Naive Bayes	0.4285	0.0186	0.7183	0.7183	0.9814	0.7139
Random Forest	0.0036	0.0176	0.9914	0.9914	0.9824	0.9879
SVM	0.0095	0.0321	0.9820	0.9820	0.9679	0.9757
DL Classifier (proposed model)	0.0047	0.0093	0.9937	0.9937	0.9907	0.9912

Table 4. Deep learning results with different combinations of hidden layers

No of layers	No of neurons	FPR	FNR	Accuracy	Precision	Recall	F1
1	50	0.0067	0.0121	0.9914	0.9914	0.9879	0.9879
1	100	0.0047	0.0093	0.9937	0.9937	0.9907	0.9912
1	150	0.0067	0.0121	0.9914	0.9914	0.9879	0.9879
1	200	0.0047	0.0139	0.9920	0.9920	0.9861	0.9888
2	100,50	0.0052	0.0111	0.9927	0.9927	0.9889	0.9898
2	100,100	0.0062	0.0149	0.9907	0.9907	0.9851	0.9870
2	200,50	0.0057	0.0102	0.9927	0.9927	0.9898	0.9898
2	200,100	0.0073	0.0121	0.9910	0.9910	0.9879	0.9875
2	200,200	0.0057	0.0139	0.9914	0.9914	0.9861	0.9879
3	100,100,50	0.0031	0.0204	0.9907	0.9907	0.9796	0.9869
3	100,100,100	0.0062	0.0176	0.9897	0.9897	0.9824	0.9856
3	200,100,100	0.0041	0.0186	0.9907	0.9907	0.9814	0.9869
3	200,200,200	0.0052	0.0111	0.9927	0.9927	0.9889	0.9898

The proposed model DL classifier is evaluated with different combination of layers and neurons using classification metrics FPR, FNR, Accuracy, recall and F1. Layer is the collection of nodes operating within a neural network. Input layer has raw data. The hidden layer in which computation is done and in output layer will have single output with multiple nodes in it. By adding more hidden layer or more neurons per layer will add more parameter to the model. In Table 4 number of neuron is increased in each layer and it is evaluated using classification metrics to know the efficiency of the proposed model.

In order to highlight the significance of the results, we compared the proposed work with other static analysis based works using the classification metrics accuracy, precision, recall and f1 score. [14] used a total of 179 static features with mixed feature set and was able to achieve an accuracy of 97.5%. [15] also uses a mixed feature set along with certificate verification and was able to achieve an accuracy of 95.31%. [16] used all the samples in Android malware genome project and was able to achieve an accuracy of 99.3% by introducing the principle of similarity to Gated Recurrent Unit. The proposed model uses DNN algorithm with total of static features Intent, API calls and permission was able to achieve an accuracy of 99.37% which is way better than other existing works and is shown in Table 5.

Table 5. Comparison of the proposed model with existing works

Related work	No of Malware/ Benign samples	Accuracy	Precision	Recall	F1
Ensemble learning [14]	2925/3938	0.9750	-	0.9730	-
DNN [15]	600/600	0.9531	0.9535	0.9531	0.9531
GRU [16]	5560/123453	0.9930	0.9879	0.9950	0.9912
DL Classifier (proposed model)	5560/9476	0.9937	0.9937	0.9907	0.9912

5.2. Performance Analysis of Second Model

The model using the static feature Dalvik Opcodes is tested with 40 applications obtained from Google Play Store and 60 applications obtained from GitHub repository. This model scrutinize the downloaded application and predicts each application whether it is benign or malware and the predictions are shown in Table 6.

Table 6. Evaluation of Models

Number of layers	Number of neurons	Benign Prediction	Malware Prediction
1	100	26/40	57/60
1	200	33/40	57/60
1	300	26/40	59/60
2	200,150	27/40	60/60
2	300,150	25/40	59/60
3	200,150,100	0/40	60/60
3	300,150,100	31/40	59/60

6. Conclusion and future work

The aim is to identify malicious functions present in an apk file of an android application and classify them as benign or malware. Two different static analysis models was trained to accomplish this task. One model classifies applications using the static features intents, API calls and permissions and the other model classifies applications using the static feature Dalvik opcodes. The future work of this paper include classifying android applications using dynamic analysis. We run the application in an emulator and classify the applications based on its behaviour. After that we will combine both static and dynamic analysis which further improves the detection accuracy. Then we will build an android application that allows the user to scan any specific application for malware by moving the apk file of the chosen application from the mobile to the system and then sending the results back to the mobile after classification. In future we will manually prepare the dataset for static analysis by installing different benign and malware applications and extracting the static features to boost the accuracy of the existing system.

References

[1] Alzaylaee MK, Yerima SY, Sezer S. DL-Droid: Deep learning based android malware detection using real devices. Computers & Security. 2020 Feb 1;89:1-25.

[2] Hota A, Irolla P. Deep Neural Networks for Android Malware Detection. InICISSP 2019 (pp. 657-663).
[3] Naway A, Li Y. Android malware detection using autoencoder. International Journal of Computer Engineering and Applications. 2019 Jan;12:1-9.
[4] Lubuva H, Huang Q, Msonde GC. A review of static malware detection for Android apps permission based on deep learning. International Journal of Computer Networks and Applications. 2019 Sep;6(5):80-91.
[5] Pan Y, Ge X, Fang C, Fan Y. A systematic literature review of android malware detection using static analysis. IEEE Access. 2020 Jun 16;8:116363-79.
[6] Chen T, Mao Q, Yang Y, Lv M, Zhu J. TinyDroid: a lightweight and efficient model for Android malware detection and classification. Mobile information systems. 2018 Oct 17;2018.
[7] SaiRamesh L, Ashok E, Sabena S, Ayyasamy A. Credit Card Fraud Detectıon in Retaıl Shopping Using Reinforcement Learning. InInternational Conference On Computational Vision and Bio Inspired Computing 2018 Nov 29 (pp. 1541-1549). Springer, Cham.
[8] Kang H, Jang JW, Mohaisen A, Kim HK. Detecting and classifying android malware using static analysis along with creator information. International Journal of Distributed Sensor Networks. 2015 Jun 3;11(6):1-9.
[9] Kang B, Yerima SY, Sezer S, McLaughlin K. N-gram opcode analysis for android malware detection. International Journal on cyber situational Awareness. 2016 Dec 5;1:231-55.
[10] Sandeep HR. Static analysis of android malware detection using deep learning. In 2019 International Conference on Intelligent Computing and Control Systems (ICCS) 2019 May 15 (pp. 841-845). IEEE.
[11] Sirisha P, Anuradha T. Detection of Permission Driven Malware in Android Using Deep Learning Techniques. In2019 3rd International conference on Electronics, Communication and Aerospace Technology (ICECA) 2019 Jun 12 (pp. 941-945). IEEE.
[12] Derbin [Internet]. [cited 2020 Oct 23]. Available from: https://www.sec.cs.tu-bs.de/danarp/derbin/
[13] Android malicious code detection based on machine learning, n-gram opcode + RandomForest [Internet]. Available from: https://github.com/kassadinsw/AndroidMalware-ngram-RF
[14] Yerima SY, Sezer S, Muttik I. High accuracy android malware detection using ensemble learning. IET Information Security. 2015 Oct 12;9(6):313-20.
[15] Naway A, Li Y. Using deep neural network for Android malware detection. International Journal of advanced studies in Computer Science and Engineering (IJASCSE). 2019 Jan 16;7(12):1-9.
[16] Zhou H, Yang X, Pan H, Guo W. An Android Malware Detection Approach Based on SIMGRU. IEEE Access. 2020 Jul 29;8:148404-10.