Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 184 (2021) 847–852

Procedia Computer Science

www.elsevier.com/locate/procedia

The 2nd International Workshop on Data-Driven Security (DDSW 2021)
March 23 - 26, 2021, Warsaw, Poland

# Android Malware Detection Using Deep Learning

Omar N. Elayan*, Ahmad M. Mustafa

*Department of Computer Information Systems , Jordan University of Science and Technology , Irbid 22110, Jordan*

**Abstract**

The Android operating system ranks first in the market share due to the system's smooth handling and many other features that it provides to Android users, which has attracted cyber criminals. Traditional Android malware detection methods, such as signature-based methods or methods monitoring battery consumption, may fail to detect recent malware. Therefore, we present a novel method for detecting malware in Android applications using Gated Recurrent Unit (GRU), which is a type of Recurrent Neural Network (RNN). We extract two static features, namely, Application Programming Interface (API) calls and Permissions from Android applications. We train and test our approach using CICAndMal2017 dataset. The experimental results show that our deep learning method outperforms several methods with accuracy of 98.2%.

*Keywords:* Android Malware; Static analysis; API-calls; Permissions; Gated Recurrent Unit.

## 1. Introduction

Nowadays, the reliance on smartphone devices is increasing dramatically more than ever before. By 2021, the number of smart mobile devices worldwide has reached 3.8 billion [1]. Moreover, more than 72% of these devices are running on the Android operating system [2]. Besides, Android users seldom install antivirus software on their devices. Even those who install it may not be able to use it very effectively to detect viruses [3]. These factors may lead to making the Android system more interesting for cyber attackers due to the large number of its users worldwide as well as the large number of valuable information that they may be able to access on these devices [4].

Notably, when the number of users increases, the amount of valuable information that can be accessed by the cyber attacker increases too. The attacker may gain access through an application which he/she has already succeeded in uploading it to Google Play. Then the victim installs it, and unknowingly gives the attacker the access [5, 6]. As of the 3rd quarter of 2020, more than 2.86 million Android mobile applications were available for download [7]. Furthermore,

* Corresponding author. Tel.: +962-788-884-445.
*E-mail address:* onelayan18@cit.just.edu.jo, ammustafa@just.edu.jo

10.1016/j.procs.2021.03.106

482,579 Malware applications were discovered every month, which means approximately 16,000 applications per day [8]. This sheer amount of Malware applications requires more sophisticated malware detection methods.

Android malware detection approaches can be divided into static, dynamic, and hybrid analysis [9]. The Static analysis extracts the features from Android application without running them into a device or Android emulator. Hence, monitoring suspicious or unusual behaviors. It can achieve high features coverage but faces many disadvantages, namely code obfuscation and dynamic code loading. On the other hand, dynamic analysis extracts the features by running them on an Android emulator or device. This approach may prove an advantage over static analysis by identifying more features or risks that may not be recognized in the static analysis. However, static analysis outperforms dynamic analysis in the cost of time and computational resources [10]. Conversely, hybrid analysis is a mixture between the two previous ones, static and dynamic. It is more effective and efficient in the detection process [11]. Our approach uses static analysis.

In essence, both of machine learning techniques and deep learning approaches have proven its effective in detecting malware in several studies, whether in Android applications analysis and the other areas of cybersecurity [12, 13, 14]. In this paper, we examine the performance of five classification algorithms; Support Vector Machine (SVM), K-Nearest Neighbors (KNN), Decision Tree (DT), Random Forest (RF) and Naive Bayes (NB), in detecting Android malware. Then, we compare its results with the deep learning GRU approach using CICAndMal2017 dataset [15]. The process of feature extraction and selection on the dataset provides both of permission and API-call features, since that malware require unusual features of permissions and API calls that are different than those of benign applications [16].

The rest of this research organized as follows; section 2 summarizes the previous literature, section 3 introduces the workflow of the research methodology, section 4 presents the results of research experiments, and the final section 5 ends with concluding the work.

## 2. Literature Review

Attackers who make malicious applications have come up with new methods of targeting victims of Android users. So, many researchers have verified the effectiveness of the available tools or suggested new tools that may be more effective in the process of accurately detecting malicious applications.

Abdulrahman et al.[17] offers a new mechanism for detecting malicious Android applications by using pseudo-dynamic analysis and constructing an API call graph for each execution path, based on a deep learning model built and trained on a data set consisting of approximately 30 thousand malicious apps and 25 thousand benign apps. However, the researchers used an API call graph to symbolize all viable execution paths that malware may track over its running time. Therefore, transform it into a low dimension numeric vector feature set to be inserted into the deep neural network. They also centered on maximizing the network efficiency by evaluating completely different embedding algorithms and tuning various network configuration parameters to guarantee that the hyper-parameters best assortment has obtained to reach the highest accuracy results. Their results are summarized as that the offered malware classification is reached at 98.86% level in accuracy, 98.65% in F-measure, and 98.47%, 98.84% within the recall and precision, respectively.

Suleiman et al. [18] proposed a classification approach based on parallel machine learning to detect Android malware. Depends on real malware samples and benign applications have derived from it, a total of 179 training features were extracted and divided into API calls and commands related: 54 features. App permissions: 125. A composite classification model was developed from a parallel set of heterogeneous classifiers, namely Simple Logistic, Naïve Bayes, Decision Tree, PART, and RIDOR. Their results are summarized that PART managed to outperform all the other classifiers, as it achieved true-positive rate 0.95%, true-negative rate 0.96%, false-positive rate 0.03%, false-negative rate 0.04%, accuracy 0.96%.

Similarly, Long et al. [19] suggested a machine learning-based lightweight system that can distinguish a benign from a malicious application. Also, they extracted features for applications using both static and dynamic methods. Moreover, they introduce a new approach proven effective in feature selection (PCA-RELIEF) to decrease the features' dimensions. Their study proved highly effective in detecting malware by using both their SVM model and the proposed new model to reduce the dimensions, where he paved the way for them to get better with a higher detection rate and lower error detection rate. All of these studies outperformed traditional methods of detecting malware in Android applications.

The difference between this paper and the rest of the research mentioned above lies in the fact that the data used are recent. Our paper presents the effectiveness of multiple traditional classification algorithms versus deep learning approaches in detecting Android malware applications.

## 3. Methodology

The schematic representation of the architecture of this work (Figure 1) shows the first phase starting with the mixture of malware and benign Android Application Package (APK) files taken from a dataset that is publicly available to everyone in [15]. Then, we extract the static features by our own Python script written in Jupyter notebook environment [20]. The features are API-calls and permissions. These features are formatted and stored in Comma-Separated Values (CSV) file as a data frame that serves the training process. Finally, we test all classifiers using 10-folds cross-validation, in addition to calculating the appropriate metrics for the evaluation.

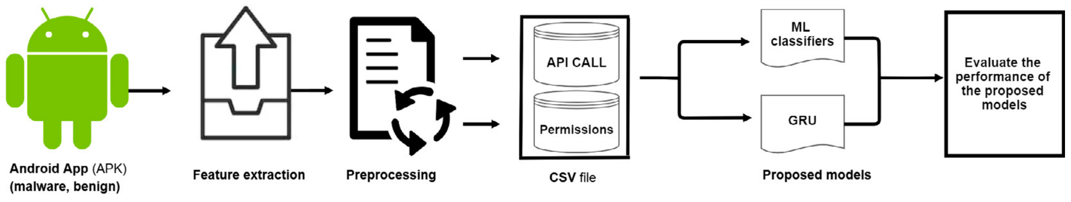

Fig. 1: Malware detection phases

### 3.1. Dataset

Indeed, there is a need for a comprehensive and reliable dataset to help deep and machine learning models test and validate the Android malware detection system. Therefore, this study has used part of the CICAndMal2017 dataset, which has been produced and published by Lashkari et al. [15], and is available on the Canadian Institute of Cybersecurity website [21]. It was gathered based on real experiments. Our dataset is a combination of 347 benign samples and 365 malware samples of Android applications, where the malwares consist of groups called families, namely; Adware, Ransomware, Scareware, and SMSware. Each family performs specific attacks that is related to its name, where the Adware and SMSware sends annoying adds and SMS during running the application, while Scareware and Ransomware ask the users for fees and ransoms so that attackers do not crash their system or stall their private data.

### 3.2. Feature Extraction and Preprocessing

In term of classification, it is essential to choose features that indicate which class the new record would belong to. From this standpoint, the permissions and API-calls are extracted from all Android applications, and both were included as features in the dataset. Androguard is a full package tool designed to interact with Android files and restricted only to python environments [22]. It can be used as a tool for reverse engineering single Android applications. The Androguard tool is used to analyze APK files by separately extracting the DEX file permissions for each APK file. Hence, we constructed a data frame containing features (columns) and applications (rows), where each column represents specific permission or API-call with binary values, while rows represent both malware and benign APK files.

### 3.3. The Classifier Framework

Traditional machine learning classifiers often show high performance in dealing with labeled data [23]. We use multiple machine learning classifiers, namely, Support Vector Machine (SVM), K-Nearest Neighbors (KNN), Decision Tree (DT), Random Forest (RF), and Naive Bayes (NB). Different metrics have been calculated to evaluate the classifiers and pick the best classifier that can give optimal results, where the evaluation involves investigating accuracy, F-measure, Recall, and Precision scores that is calculated based on the below equations.

$$Precision = \frac{TruePositive}{(TruePositve + FalsePositive)} \qquad Recall = \frac{TruePositive}{(TruePositve + FalseNegative)}$$

$$Accuracy = \frac{TP + TN}{(TP + TN + FP + FN)} \qquad F - Measure = \frac{2 \times (Precision \times Recall)}{Precision + Recall}$$

Accuracy is a metric for evaluating classification models, which is the fraction of predictions that the model obtained correctly. Precision is the number of appropriately recognized positive outcomes divided by the variety of all positive results, together with these not recognized correctly. Also, the Recall is the variety of properly-recognized outcomes divided by the variety of all samples that should have been appropriately recognized. As well as the Precision can be thought of as a measure of a classifier exactness, where low Precision indicates a large number of false positives. Recall measures the completeness of classifiers, where low Recall indicates many false negatives. F-Measure conveys the balance between Precision and Recall, where it can be used to select the model based on a balance between Precision and Recall [24].

### *3.4. GRU Architecture*

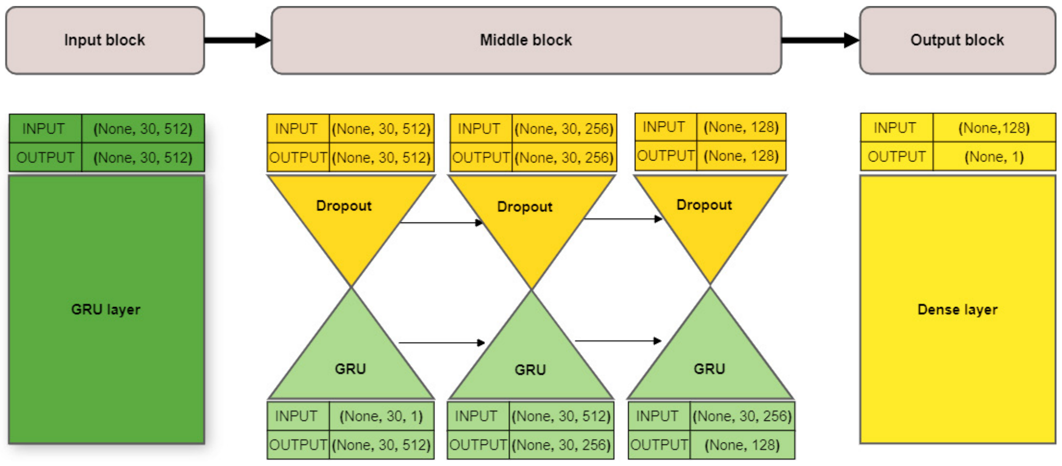

Fig. 2: Architecture of the deep learning model

We use GRU in our approach to build a binary classification model. The architecture is constructed as shown in Figure 2. The model consists of three blocks; input block, middle block, and output block. The input block contains GRU input layer, while the middle block contains three GRU layers with (512, 256, and 128) neurons and each one is connected with dropout layer of 0.2 dropout rate for regularization propose. Finally, the output block contains a dense layer that is activated by Sigmoid function for binary classification. We used the most common optimizer (Adam) with a 0.0001 learning rate. We have used the early stopping and checkpoint techniques with patience that is ten percent of the epochs to monitor loss and accuracy values. These techniques are efficient methods to prevent falling in over-fitting and under-fitting problems [25].

## 4. Experimental Results

We have conducted our experiments on two types of models, traditional machine learning classifiers and deep learning. First, we train the classifiers using the dataset, and then we perform the testing and evaluation. In both experiments, the classifications are based on the features extracted from permissions and API-calls.

### 4.1. Traditional Machine Learning

Specifically, we have chosen SVM, KNN, DT, RF, and NB to build a useful model in detecting malware applications, taking into consideration that all algorithms are considered of the supervised learning. All of them are easy to implement and apply, and can assist with both classification and regression tasks. Table 1 shows the classifiers performance in detecting harmful Android applications.

Table 1: The performance of the tested algorithms in detecting Android malware.

| Metrics | SVM | KNN | DT | RF | NB | GRU |
|---|---|---|---|---|---|---|
| Accuracy | 96.2% | 97.2% | 96.6% | 97.8% | 93.9% | 98.2% |
| Precision | 98.1% | 98.3% | 96.2% | 98.8% | 94.3% | 96.9% |
| Recall | 94.4% | 96.0% | 97.3% | 97.2% | 93.5% | 99.2% |
| F-measure | 96.2% | 97.1% | 96.6% | 97.9% | 93.9% | 98.0% |

RF (Random Forest) classifier has achieved the highest level of accuracy with 97.8% score, and the highest score of correct predictions, where the F-score showed that it was able to predict 97.9% of the dataset correctly. Also, RF classifier has achieved the second level of recall, in which it was able to predict 97.2% of the malware samples, correctly. Regardless to that, RF classifier has achieved the highest level of precision, in which it was able to identify 98.8% of the whole dataset during testing process.

### 4.2. Deep Learning

Our deep learning approach have classified Android based on permissions and API calls. We have chosen the hyper-parameters based on the accuracy and loss metrics.

We stopped the model training at the 25th Epoch, since the loss rate in training and testing have become 0.07%, and the level of accuracy have reached 98.2%. Also, deep learning classifier was able to correctly predict 98.0% of the dataset. Furthermore, it has achieved high score of recall and precision, in which it was able to correctly detect 99.2% of the malware samples.

The results show that the model performance was enhanced using deep learning approaches, where the results of deep learning classifier was better than the results of the traditional machine learning classifiers. But, both experiments have achieved excellent result, thus the models that are based on permissions and API-calls considered to give promising results in term of Android malware detection.

## 5. Conclusion

This research has introduced a new model for detecting malware in Android OS applications by building GRU architecture of Recurrent Neural Network (RNN) deep learning approaches. Moreover, presenting a comparison between the traditional machine learning techniques and the deep learning approaches, in order to choose the most efficient model that achieves the highest results in detecting Android malware. The proposed classifiers were trained on a dataset that was taken from the CICAndMal2017 dataset, noting that the dataset was analyzed by a static analysis of real and realistic samples of malware and benign Android applications. The analysis was for both permissions and the API-calls, knowing that using both showed that it was worth relying more on detecting Android malware models. The deep learning classifier has outperformed all other classifiers, reaching a 98.2% level of accuracy and 98.0% score of F-measure using permissions and API-calls static features in Android malware detection.

## References

[1] Number of Smartphone and Mobile Phone Users Worldwide in 2020/2021: Demographics, Statistics, Predictions. https://financesonline.com/number-of-smartphone-users-worldwide/. Accessed: 2020-Jan-11.

[2] Mobile Operating System Market Share Worldwide 2020. https://gs.statcounter.com/os-market-share/mobile/worldwide. Accessed: 2020-Jan-11.

[3] Rafael Fedler, Marcel Kulicke, and Julian Schütte. An antivirus api for android malware recognition. In *2013 8th International Conference on Malicious and Unwanted Software:" The Americas"(MALWARE)*, pages 77–84. IEEE, 2013.

[4] Ali Dehghantanha and Katrin Franke. Privacy-respecting digital investigation. In *2014 Twelfth Annual International Conference on Privacy, Security and Trust*, pages 129–138. IEEE, 2014.

[5] C Chia, K-KR Choo, and D Fehrenbacher. How cyber-savvy are older mobile device users? In *Mobile Security and Privacy*, pages 67–83. Elsevier, 2017.

[6] Nicolas Viennot, Edward Garcia, and Jason Nieh. A measurement study of google play. In *The 2014 ACM international conference on Measurement and modeling of computer systems*, pages 221–233, 2014.

[7] Google Play: number of available apps by quarter 2020 — Statista. https://www.statista.com/statistics/289418/number-of-available-apps-in-the-google-play-store-quarter. Accessed: 2020-Jan-12.

[8] Global Android malware volume 2020 — Statista. https://www.statista.com/statistics/680705/global-android-malware-volume/. Accessed: 2020-Jan-12.

[9] Kaijun Liu, Shengwei Xu, Guoai Xu, Miao Zhang, Dawei Sun, and Haifeng Liu. A review of android malware detection approaches based on machine learning. *IEEE Access*, 8:124579–124607, 2020.

[10] Krishna Sugunan, T Gireesh Kumar, and KA Dhanya. Static and dynamic analysis for android malware detection. In *Advances in Big Data and Cloud Computing*, pages 147–155. Springer, 2018.

[11] Mahima Choudhary and Brij Kishore. Haamd: hybrid analysis for android malware detection. In *2018 International Conference on Computer Communication and Informatics (ICCCI)*, pages 1–4. IEEE, 2018.

[12] Ambra Demontis, Marco Melis, Battista Biggio, Davide Maiorca, Daniel Arp, Konrad Rieck, Igino Corona, Giorgio Giacinto, and Fabio Roli. Yes, machine learning can be more secure! a case study on android malware detection. *IEEE Transactions on Dependable and Secure Computing*, 2017.

[13] ElMouatez Billah Karbab, Mourad Debbabi, Abdelouahid Derhab, and Djedjiga Mouheb. Maldozer: Automatic framework for android malware detection using deep learning. *Digital Investigation*, 24:S48–S59, 2018.

[14] Leong Chan, Ian Morgan, Hayden Simon, Fares Alshabanat, Devin Ober, James Gentry, David Min, and Renzhi Cao. Survey of ai in cybersecurity for information technology management. In *2019 IEEE Technology & Engineering Management Conference (TEMSCON)*, pages 1–8. IEEE, 2019.

[15] Arash Habibi Lashkari, Andi Fitriah A Kadir, Laya Taheri, and Ali A Ghorbani. Toward developing a systematic approach to generate benchmark android malware datasets and classification. In *2018 International Carnahan Conference on Security Technology (ICCST)*, pages 1–7. IEEE, 2018.

[16] Naser Peiravian and Xingquan Zhu. Machine learning for android malware detection using permission and api calls. In *2013 IEEE 25th international conference on tools with artificial intelligence*, pages 300–305. IEEE, 2013.

[17] Abdurrahman Pektaş and Tankut Acarman. Deep learning for effective android malware detection using api call graph embeddings. *Soft Computing*, 24(2):1027–1043, 2020.

[18] Suleiman Y Yerima, Sakir Sezer, and Igor Muttik. Android malware detection using parallel machine learning classifiers. In *2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies*, pages 37–42. IEEE, 2014.

[19] Long Wen and Haiyang Yu. An android malware detection system based on machine learning. In *AIP Conference Proceedings*, volume 1864, page 020136. AIP Publishing LLC, 2017.

[20] Python script for feature extraction in each API and permission, static analysis for Android malware detection in Machine learning . https://github.com/OmarElayan96/Android_malware/tree/main. Accessed: 2020-Jan-14.

[21] Number of Smartphone and Mobile Phone Users Worldwide in 2020/2021: Demographics, Statistics, Predictions. https://www.unb.ca/cic/datasets/andmal2017.html. Accessed: 2020-Jan-11.

[22] Androguard documentation. https:https://github.com/androguard/androguard/blob/master/docs/index.rst. Accessed: 2020-Jan-22.

[23] Peter Harrington. *Machine learning in action*. Manning Publications Co., 2012.

[24] David MW Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. *arXiv preprint arXiv:2010.16061*, 2020.

[25] Sunila Gollapudi. *Practical machine learning*. Packt Publishing Ltd, 2016.