engineering proceedings

MDPI

Proceeding Paper

Feature-Based Semi-Supervised Learning Approach to Android Malware Detection †

Mariam Memon [1,*], Adil Ahmed Unar [1], Syed Saad Ahmed [2], Ghulam Hussain Daudpoto [1] and Rabeea Jaffari [1,*]

[1] Software Engineering Department, Mehran University of Engineering and Technology, Jamshoro 76062, Pakistan; adilahmedaptech@gmail.com (A.A.U.); ghullamhussaindpo@gmail.com (G.H.D.)
[2] Computer Systems Engineering Department, Mehran University of Engineering and Technology, Jamshoro 76062, Pakistan; saadjaff15@gmail.com
* Correspondence: mariam.jawaid@faculty.muet.edu.pk (M.M.); rabeea.jaffari@faculty.muet.edu.pk (R.J.)
† Presented at the 2nd International Conference on Emerging Trends in Electronic and Telecommunication Engineering, Karachi, Pakistan, 15–16 March 2023.

Abstract: The development of signature-based methods or Machine Learning (ML) techniques on static data has dominated automated malware detection on android platforms. However, these techniques may not detect dangerous activities that only manifest during runtime. Furthermore, there is already a significant volume of unlabeled malware data available, making the production of datasets through supervised ML approach of manual labelling expensive. For anti-virus researchers, the process of malware development poses a significant engineering challenge because they lack an effective method for capturing potentially new harmful files while removing clean and well-known files. In this research, we propose a semi-supervised ML technique to detect android malware from android permissions and Application Programmer Interface (API) call logs. The ML technique is incorporated into an android application to scan the installed applications and detect the corresponding levels of maliciousness with success. The results depict the feasibility of our proposed method and application.

Keywords: malware detection; android malware; static analysis; machine learning; semi-supervised learning

Citation: Memon, M.; Unar, A.A.; Ahmed, S.S.; Daudpoto, G.H.; Jaffari, R. Feature-Based Semi-Supervised Learning Approach to Android Malware Detection. *Eng. Proc.* **2023**, *32*, 6. https://doi.org/10.3390/engproc2023032006

Academic Editors: Muhammad Faizan Shirazi, Saba Javed, Sundus Ali and Muhammad Imran Aslam

Published: 20 April 2023

1. Introduction

The past decade has experienced an increasing number of mobile devices with android being the most popular operating system for these devices [1]. Android devices have increased in quantity from merely 38 in 2009 to an overwhelming number of over 20,000 devices in 2016 [2]. The popularity and pervasiveness of android devices makes them an attractive target for malicious offenders. The more these devices grow, the more we have experienced the growth of Android malware. As per the reports from Statista, Android malware has increased to 26.6 million in March 2018 [3]. Similar reports from G Data confirm that Android malware has reached 3.2 million and it increased by 40% year-on-year in the third quarter of 2018 [4]. Android malware is hidden inside various applications available in the Android market and gets installed on an individual's Android device without any explicit permission.

Android malware not only threatens the end user's privacy, but also lessens the trust on security policies of Android devices. The typical behavior of these malicious applications includes stealing and modifying user information, disabling a mobile device, maliciously controlling the mobile device, browser hijacking, and so on [5,6]. The threats posed by Android malware can be addressed via three major malware analysis techniques namely the static, dynamic, and hybrid analysis techniques. Static techniques detect malware prior to the application installation by scanning and traversing all possible execution paths of the android package kit (APK) to identify malware signatures. Thus, these techniques detect

malware quickly without running the android application [7]. On the other hand, Dynamic techniques detect malware by executing the APK [8]. Compared to static techniques, dynamic techniques are more time-consuming, resource intensive, and fail to identify the parts of code outside the monitoring range's execution [9]. Although scalable, the static techniques also suffer from certain issues, such as java reflection and dynamic code loading [10]. Hybrid analysis techniques integrate static and dynamic techniques to improve the malware detection accuracy, but result in a waste of space and time to detect and analyze the huge number of malware samples [11]. Major problems pertaining to all the discussed traditional detection techniques are that either these can only detect malware with known signatures (static) or either the scanning process to detect malwares is a time and resource consuming task (hybrid and dynamic).

Researchers and practitioners are using ML to successfully detect android malware. Contrary to the traditional detection practices, ML can detect more sophisticated malware with unknown signatures and reduces the time-consuming process of traditional analysis methods [12]. ML-based Android malware detection works by training a ML model on a set of features, then utilizing the trained model to detect the malware. Features employed to train the ML models for Android malware detection include permissions, API call logs, network addresses, code metrics, malware signatures, and so on [13]. The ML techniques can broadly be divided into three categories, namely, supervised, unsupervised, and semi-supervised techniques. The supervised techniques need labelled data for accurate detection while the unsupervised techniques model a set of inputs without any labelled data, but yield lower performances than supervised methods. Semi-supervised techniques combine the supervised and unsupervised techniques to yield an optimal detector without huge amount of labelled data [14].

In this research, we propose a semi-supervised based ML technique to Android malware detection. Our method takes the permission and component information along with the API call logs of the android application as features to train a semi-supervised Naïve Bayes classifier [15] algorithm to detect android malware. These features are extracted from a set of labelled (malware and genuine software) and unlabeled examples to create an ML Naïve Bayes classifier. The proposed ML method is an advancement in Android malware detection to detect sophisticated malware with unknown signatures efficiently without the use of excessively labelled data.

The rest of the paper is organized as follows: Section 2 presents the methodology to implement the proposed semi-supervised ML technique for efficient Android malware detection; Section 3 discusses the results of the proposed method followed by conclusion and future work at the end.

2. Methodology

The methodology for the proposed feature-based semi-supervised ML approach to android malware detection is depicted in Figure 1.

As seen from Figure 1, the proposed technique has two data sources: Google Play Store for benign applications, and Virus Total for malicious applications. The APK files collected from these sources are run on a sandbox of emulators to extract meaningful features to distinguish them. Once the dataset is created, it is used for training our model using a semi-supervised approach. This model becomes the base of our android malware detection application. Various mechanisms employed for the implementation of the proposed technique are discussed in the subsequent subsections.

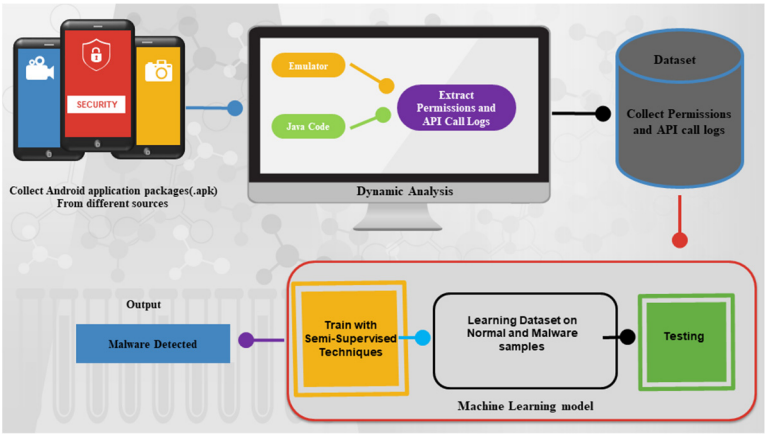

Figure 1. Methodology for the proposed feature-based semi-supervised learning approach to android malware detection.

2.1. Android Application Packages Collection

A total of 54,332 android applications were collected from both the Google Play Store and Virus Total in equal proportion to create a dataset of benign and malware applications, respectively. These applications were then distributed into 31 categories based on the type of applications they were, such as games, e-commerce, educational, and so on.

2.2. Extracting Permissions and API Call Logs

The features space for the malware detection approach used in this work constitutes of 1488 permissions and API call logs. These features are extracted by executing these applications on emulators and saved in a CSV file in accordance with the category they belonged to. The filtered API calls made while running the APK files are included in the behavior data, and permissions are added as static data. A binary feature vector indicating the presence of an API request or permission is used to represent each APK. For example, if an APK is represented by X, then the feature vector associated with it will have the following pattern (1, 0, . . . 0), where 1 indicates the existence of a particular API call or permission and 0 otherwise. These features constitute the dataset employed for training the ML model in the next step.

2.3. ML Model Training and Performance Evaluation

The finalized dataset was split into training and test sets in 80–20% ratio using Sci-Kit Learn library. The training set was then utilized to train the semi-supervised naïve Bayes (NB) classifier ML model. The performance of this model was evaluated against other state-of-the-art ML models: K-Neighbor (KNN) Classifier [16], Decision Trees [17], Random Forest (RBF) [18], and Support Vector Machines (SVM) [19], which were also trained using the same training set. The 20% test set is used to evaluate the performance of all the ML models as mentioned in Section 4 ahead. All models were trained using Python TensorFlow [20].

2.4. Malware Detection Application Implementation

The Malware detection App is developed using Java and Python as its central programming languages. Android Studio IDE is employed for front-end development whereas TensorFlow [20], Sci-Kit Learn [21], Jupyter notebook [22], and Keras [23] are utilized for the back end.

To identify fraudulent applications, the application makes use of API calls logs and permissions. While scanning the device, it extracts permissions and API call logs of each installed application and loads the trained Naïve Bayes (NB) ML model. A vector representing these extracted features is subsequently supplied to our model which yields a prediction score (ps) between 0 and 1 representing whether an application is trustworthy, malicious,

risky, or unknown. After extensive research, we identified the ranges of prediction scores to classify the applications into different levels of maliciousness. These levels are specified in Table 1.

Table 1. Categorization of applications based on the prediction scores from the trained ml model.

S.#	Prediction Score (ps)	Maliciousness Level
1	$ps \leq 0.5$	Safe/Goodware
2	$0.5 > ps < 0.75$	Risky

It is noteworthy that if the Malware detection App is unable to extract permissions and API calls from any application then the application is labelled as 'Unknown' and its prediction score is not calculated. Figure 2 shows the summarized workflow of our Malware Detection App.

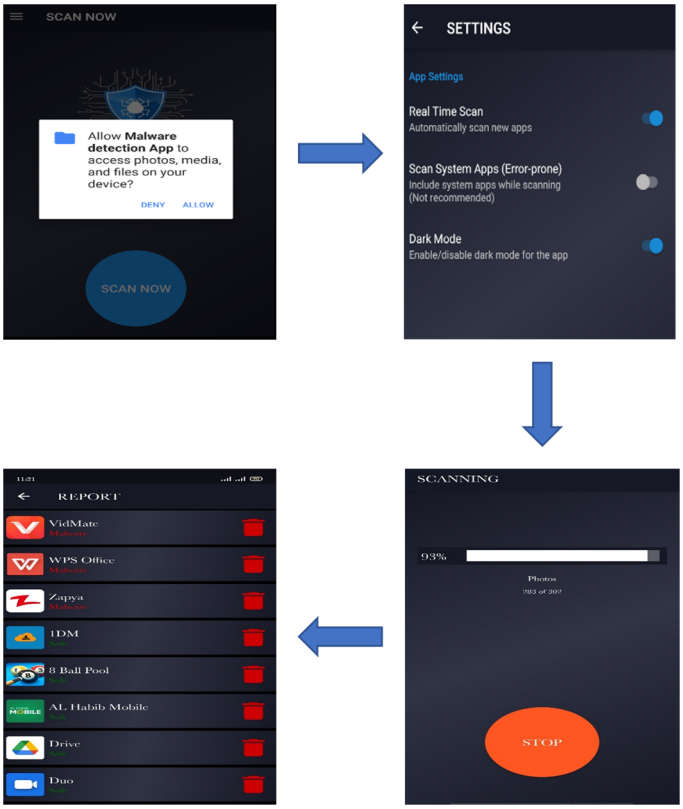

Figure 2. Summarized workflow of the Malware Detection App incorporating the proposed feature-based semi supervised learning approach to android malware detection.

3. Results and Discussion

The results section is divided into two parts: (1) Performance evaluation of the proposed feature-based semi supervised learning approach to android malware detection against other state-of-the-art ML models, and (2) Malware detection results from the Malware Detection App.

3.1. ML Model Performance Evaluation Results

As already mentioned in Section 3, the proposed semi-supervised ML model based on Naïve Bayes (NB) classifier is evaluated against four other (K-Neighbor (KNN) Classifier, Decision Trees, Random Forest (RBF), and Support Vector Machine (SVM)) state-of-the-art

ML models using the test set and the results are depicted in Figure 3 using the Accuracy measure, which is defined in (1).

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions} \quad (1)$$

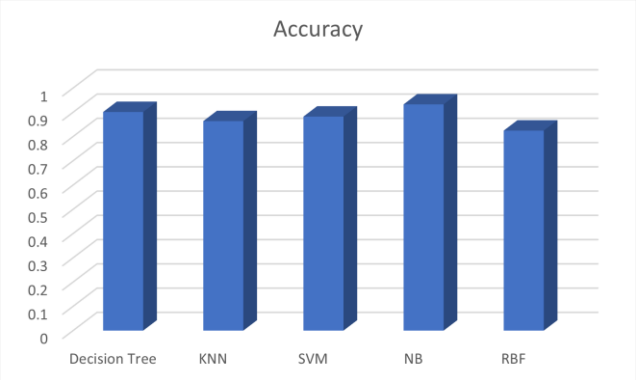

Figure 3. Performance evaluation of the proposed semi-supervised Naïve Bayes (NB) classifier against four other (K-Neighbor (KNN) Classifier, Decision Trees, Random Forest (RBF), and Support Vector Machine (SVM)) state-of-the-art ML models on test set.

As depicted in Figure 3, our proposed Naïve Bayes (NB) classifier yields the best performance with 93.256% accuracy on the test set, followed by Decision Trees with second best performance of 90.124%, while the Support Vector Machines (SVM), Random Forest (RBF), and K-Nearest Neighbors (KNN) yield accuracies of 88.227%, 82.453%, and 86.354%, respectively. These results depict the superiority of the Naïve Bayes (NB) semi-supervised technique on the android malware detection test set using the permission and API call log features.

3.2. Malware Detection App Results

The Malware Detection App based on the trained NB classifier scans the installed android applications for malware and predicts a score (between 0–1) and the level of maliciousness (as specified in Table 1). The results from the application are depicted in Figure 4.

Figure 4 depicts that the app successfully detects various applications as 'safe' or 'malware' based on the prediction score received from the trained ML model with the range decided via Table 1.

Figure 4. Malware app detection results specifying the levels of maliciousness.

4. Conclusions and Future Work

In this paper, we proposed a semi-supervised ML android malware detection technique using a combination of labelled and unlabeled permissions and API call logs data to detect malware applications on an Android device. Our technique yielded the best accuracy of 93.256% when compared against other state-of-the-art ML models. The proposed technique was employed to implement an android application (Malware Detection App) to scan and classify various applications into different levels of maliciousness according to a prediction score received via the proposed ML technique. The detection results strengthen our confidence in the use of semi-supervised malware detection for anti-malware research.

As far as future work is concerned, various other features apart from the application permissions and API call logs can be utilized to train the semi-supervised ML model for android malware detection.

Author Contributions: Conceptualization, M.M. and A.A.U.; methodology, A.A.U. and G.H.D.; software, A.A.U. and G.H.D.; validation, R.J. and S.S.A.; formal analysis, S.S.A.; investigation, G.H.D. and S.S.A.; data curation, S.S.A.; writing—original draft preparation, M.M. and R.J.; writing—review and editing, R.J.; visualization, R.J. and S.S.A.; supervision, M.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Jkielty. Android v iOS Market Share. 2019. Available online: https://deviceatlas.com/blog/android-v-ios-market-share (accessed on 20 July 2022).
2. Android. What Is Android. 2019. Available online: https://www.android.com/what-is-android/ (accessed on 1 August 2022).
3. Statista. Development of New Android Malware Worldwide from 2011 to 2018. 2019. Available online: https://www.statista.com/statistics/680705/global-android-malwarevolume/ (accessed on 15 July 2022).
4. Data, G. The Number of New Malicious Android Samples Worldwide in Q3 2018 Reached 3.2 Million, an Increase of 40% Year-On-Year. 2018. Available online: http://www.199it.com/archives/793849.html (accessed on 15 August 2022).
5. Zhou, Y.; Jiang, X. Dissecting android malware: Characterization and evolution. In Proceedings of the 2012 IEEE Symposium on Security and Privacy, San Francisco, CA, USA, 20–23 May 2012; pp. 95–109.
6. B. V. Mobile Malware. 2013. Available online: https://www.webopedia.com/definitions/mobile-malware/ (accessed on 15 August 2022).
7. Castillo, C. Android Malware Past, Present, and Future. McAfee White Paper, Mobile Security Working Group, ed. 2011. Available online: http://www.mcafee.com/us/resources/white-papers/wpandroid-malware-past-present-future.pdf (accessed on 15 July 2022).
8. Sugunan, K.; Kumar, T.G.; Dhanya, K. Static and dynamic analysis for android malware detection. In *Advances in Big Data and Cloud Computing*; Springer: Singapore, 2018; pp. 147–155.
9. Enck, W. Defending users against smartphone apps: Techniques and future directions. In Proceedings of the International Conference on Information Systems Security, Kolkata, India, 15–19 December 2011; pp. 49–70.
10. Pan, Y.; Ge, X.; Fang, C.; Fan, Y. A systematic literature review of android malware detection using static analysis. *IEEE Access* **2020**, *8*, 116363–116379. [CrossRef]
11. Fang, Y.; Gao, Y.; Jing, F.; Zhang, L. Android malware familial classification based on dex file section features. *IEEE Access* **2020**, *8*, 10614–10627. [CrossRef]
12. Ahvanooey, M.T.; Li, Q.; Rabbani, M.; Rajput, A.R. A survey on smartphones security: Software vulnerabilities, malware, and attacks. *arXiv* **2020**, arXiv:2001.09406.
13. Jusoh, R.; Firdaus, A.; Anwar, S.; Osman, M.Z.; Darmawan, M.F.; Razak, M.F.A. Malware detection using static analysis in Android: A review of FeCO (features, classification, and obfuscation). *PeerJ Comput. Sci.* **2021**, *7*, e522. [CrossRef] [PubMed]
14. Ayodele, T.O. Types of machine learning algorithms. *New Adv. Mach. Learn.* **2010**, *3*, 19–48.
15. Webb, G.I.; Keogh, E.; Miikkulainen, R. Naïve Bayes. *Encycl. Mach. Learn.* **2010**, *15*, 713–714.
16. Peterson, L.E. K-nearest neighbor. *Scholarpedia* **2009**, *4*, 1883. [CrossRef]
17. Kingsford, C.; Salzberg, S.L. What are decision trees? *Nat. Biotechnol.* **2008**, *26*, 1011–1013. [CrossRef] [PubMed]

18. Rigatti, S.J. Random forest. *J. Insur. Med.* **2017**, *47*, 31–39. [CrossRef] [PubMed]
19. Noble, W.S. What is a support vector machine? *Nat. Biotechnol.* **2006**, *24*, 1565–1567. [CrossRef] [PubMed]
20. Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G.S.; Davis, A.; Dean, J.; Devin, M.; et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv* **2016**, arXiv:1603.04467.
21. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; et al. Scikit-learn: Machine learning in Python. *J. Mach. Learn. Res.* **2011**, *12*, 2825–2830.
22. Kluyver, T.; Ragan-Kelley, B.; Pérez, F.; Granger, B.E.; Bussonnier, M.; Frederic, J.; Kelley, K.; Hamrick, J.B.; Grout, J.; Corlay, S.; et al. *Jupyter Notebooks-A Publishing Format for Reproducible Computational Workflows*; IOS Press: Amsterdam, The Netherland, 2016; Volume 2016.
23. Gulli, A.; Pal, S. *Deep Learning with Keras*; Packt Publishing Ltd.: Birmingham, UK, 2017.