Generative Ensemble Learning for Mitigating Adversarial Malware Detection in IoTs

Usman Ahmed[1], Jerry Chun-Wei Lin[1,*], and Gautam Srivastava[2]
[1]Western Norway University of Applied Sciences, Bergen, Norway
[2]Brandon University, Brandon, Canada
Email: Usman.Ahmed@hvl.no, jerrylin@ieee.org, SRIVASTAVAG@brandonu.ca

***Abstract*—This paper proposes a framework that can be employed to mitigate adversarial evasion attacks on Android malware classifiers. It extracts multiple discriminating feature subsets from a single Android app such that each subset has the potential to classify a huge dataset of malicious and benign Android apps independently. Moreover, it incorporates an ensemble of ML classifiers where each classifier is trained on different features subset. Finally, the ensemble model formulates a collaborative classification decision that is resilient against adversarial evasion attacks. Results showed that the designed model achieves good performance compared to the existing models.**

***Index Terms*—adversarial evasion attacks, ML-based ensemble analysis, ransomware detection.**

I. Introduction

Nowadays, smartphones have become an integral part of our lives as they are used in almost every domain (i.e., banking, social networking, and shopping) and applied on Internet of Things (IoT) environments to collect data for further analysis. With the mainstream usage of the Android platform on smartphones[1], McAfee reported that there are more than 12 million Android malware samples[2]. To counter these malicious activities, traditional approaches such as antivirus have been used that highly rely on the repository of malicious application signatures.

According to a previous study [20], Android malware is swiftly advancing to evade signature-based approaches and thus requires the development of more robust anti-malware solutions for smartphones. In this perspective, machine learning (ML)-based solutions are employed to characterize Android malware. In the perspective of the advances in ML-based approaches in the previous decade, the research community has shown a dominant interest in applying these to the Android malware detection [3], [18]. In most of the previous approaches [11], [15], researchers have extracted multiple features from Android apps using either static or dynamic analysis. Multiple features like APIs, permissions, intents, network addresses etc., can be extracted from an Android APK file and embedded into a single feature vector space to apply ML-based algorithms to classify malicious and benign apps. However, surprisingly, these approaches can easily be misled by using adversarial examples [2]. This introduces a challenging issue as ML theory takes the premise that the training dataset utilized in a learning phase stays illustrative of the problem domain and expects that purposefully unsafe modifications of the data do not occur [9].

In this work, our concern is to deal with adversarial evasion attacks. We then propose a novel and salable countermeasure for adversarial evasion attacks on ML-based Android malware classifiers. In addition, we identify and rank the most discriminating feature subsets extracted from Android apps for malware detection. The semantic subsets from the original feature vector and rank them according to detection accuracy are then defined. We also use the best possible ML-based classifiers with optimal hyperparameter values. Finally, the extracted discriminating feature subsets are used to train the model. Results showed that the designed model can achieve good performance than the existing approaches.

II. Related Work

Min Zheng *et al.* [23] proposed a signature-based technique called DroidAnalytics to classify malware. Similarly, Jehyun Lee *et al.* [10] developed a signature-based technique used to detect variants of known malware families. Peiravian *et al.* [17] used permissions in combination with APIs-based features and used machine learning (ML) to distinguish between malicious and benign apps. In DroidAPIminer [1] frequency analysis of API calls within benign and malware apps are made and evaluated on different classifiers using the generated feature set.

Ali Feizollah *et al.* [7] used Android intents as a feature to analyze Android malware. Moreover, Sheen *et al.* [19] employ a multi-featured collaborative decision fusion approach to detect Android malware. As discussed earlier, ML techniques are extensively applied in the classification of malware. Thus their reliability is of crucial concern. Many techniques have been adopted by the research community to mitigate adversarial attacks not only restricted to computer vision but also natural language processing and cybersecurity applications like intrusion detection systems and malware detection [5], [12], [21].

Given Android malware, ADAM [22] was the pioneer to introduce malware evasion techniques. Emre and Sevil [4] incorporated genetic programming to obfuscate Android

[1]https://www.idc.com/promo/smartphone-market-share/os
[2]https://www.mcafee.com/content/dam/consumer/en-us/docs/2020-Mobile-Threat-Report.pdf

malware application. Guozhu *et al.* [13] employed a technique to generate malware with specific features automatically. Alejandro *et al.* [6] proposed LagoDroid, a framework to trick the RevealDroid [8] classifier into miss-classifying the malware family. Moreover, the authors in [6] also proposed a countermeasure for evasion attacks caused by LagoDroid.

III. Dataset and Feature Extraction

In this study, we use Drebin dataset[3] as a benchmark. To balance the benign and malicious data in the repository, we randomly select 5,450 benign apps from a total of 213,453 benign apps in the data set. To extract static features from the data set, we reverse engineer the samples in the repository. Fig. 1 depicts the process to reverse engineer an Android app. An overview of features extracted from the Drebin dataset is listed in Table III.

TABLE I
Overview of feature subsets

Feature sets	
manifest file	Requested permissions
	Hardware components
	Filtered intents
source code	API calls
	Network addresses

For the features extraction from manifest file, Android *manifest.xml* file contains the metadata that every Android app needs at the installation time and further used while executing the application. We used the Drebin dataset to extract the following classes of features from the manifest file. Besides, Android permission is a framework to protect user privacy. Apps must request permission from the user before accessing sensitive data such as sending SMS or accessing contacts. It has been observed that the pattern of requested permissions can help identify malware. Thus, permissions prove to be a helpful feature for malware detection in Android apps.

For the filtered intents, the intent in Android is a mechanism by which different App components communicate with each other. Moreover, intents are used to transfer data from one activity to another. Information about intents is listed in the manifest file and could be used as a potential feature for malware detection. Considering the hardware components, AndroidManifest.xml also lists the hardware components which the App wants to access, such as camera, GPS or touch screen. Features based on hardware components can be helpful in the detection of malware as it may need to access a specific pattern of hardware components to perform malicious activity.

In addition, the disassembled code can be useful to extract multiple static features from the App. We use the following features from disassembled code for malware detection: (1) **API Calls:** APIs are a set of rules or specifications that an Android app needs to follow while communicating with other App components, such as sending SMS or accessing user location. Specific patterns of API calls extracted from an android application can potentially lead to malware detection. We use information about API calls in an application as a feature class to detect malware; (2) **Network Addresses:** malware often establishes connections to remote servers using specific network addresses, i.e., IP address or a domain name. Thus, we extract the network address from disassembled code to construct a feature vector that can help identify malware.

[3]https://www.sec.cs.tu-bs.de/~danarp/drebin/

IV. Feature Subsets Selection and Ranking

Our target is to employ an ensemble of classifiers-based models where each classifier in the model is trained on different subsets of features. The first step of our methodology is to investigate whether the extracted feature subsets can classify a huge dataset as malicious or benign independently. We employ a comprehensive static analysis of Android apps to extract multiple types of static features. The extracted features are further embedded in multiple feature vector spaces according to the class of features, i.e., Permissions, APIs, Intents, Hardware Components and Network addresses. After extracting the features in multiple subsets, we apply ML-based algorithms with optimal hyperparameter settings for each feature vector separately to extract the best classification results. Moreover, we rank the feature subsets based on detection accuracy. Finally, we use the selected feature subsets to design an ensemble classification model to mitigate the effects of adversarial evasion attacks.

A. Model selection

In this research, we use TPOT [14], an automated ML (AutoML) tool which designs and optimizes the ML pipelines. TPOT incorporates all the algorithms in SciKit-Learn [16] package, which is an open-source ML library for Python programmers. Therefore, each of the operators in the pipeline of the TPOT library corresponds to the particular ML algorithm for classification, algorithm for feature pre-processing and algorithm for feature selection.

B. Model training and testing

After selecting the best classification model and optimal hyperparameters, the next step involves training and testing our model for each extracted feature subset repository. We use a total of 11,010 Android apps ($5,560$ malicious and $5,450$ benign) from the benchmark Drebin dataset for training and testing by using TPOT. We distributed 70% (i.e., $7,707$) for training and 30% (i.e., $3,303$) apps for testing purpose. However, in the network address class, we could only find $3,888$ malicious samples out of $,5560$, which had URL-based features present in them. Therefore, we trained and tested our model on $9,338$ samples ($3,888$ malicious and $5,450$ benign) for the network addresses class.

Fig. 2 presents the result of the tree-based pipeline (TPoT) for permissions-based features class. Our classification results show that, among all the static features in the Android App, the API-based class has the most discriminating features for malware detection. In comparison with permissions-based features class, API-based features class has slightly less accuracy

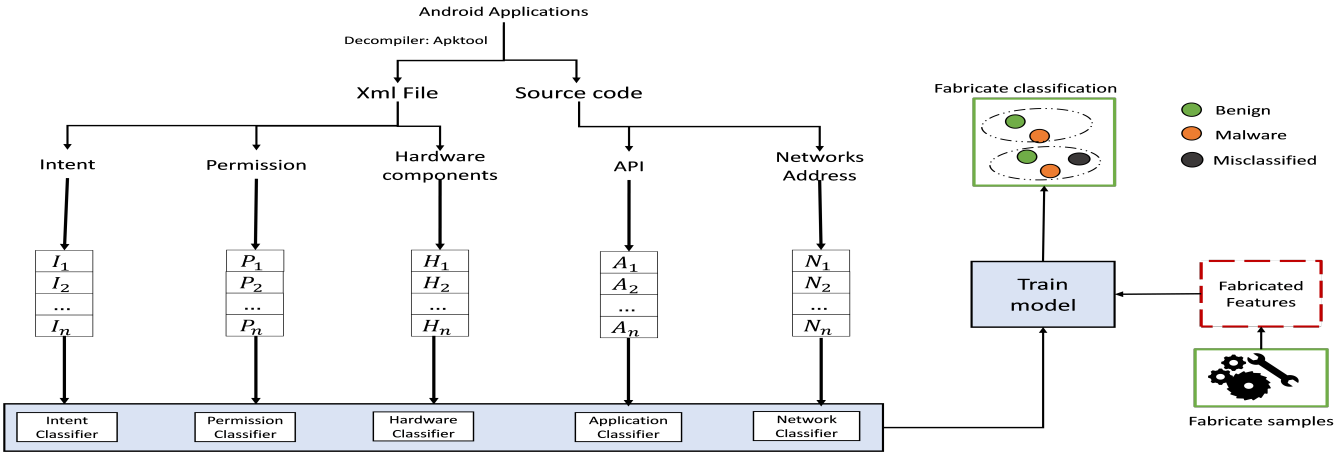

Fig. 1. Proposed model training and feature extraction method.

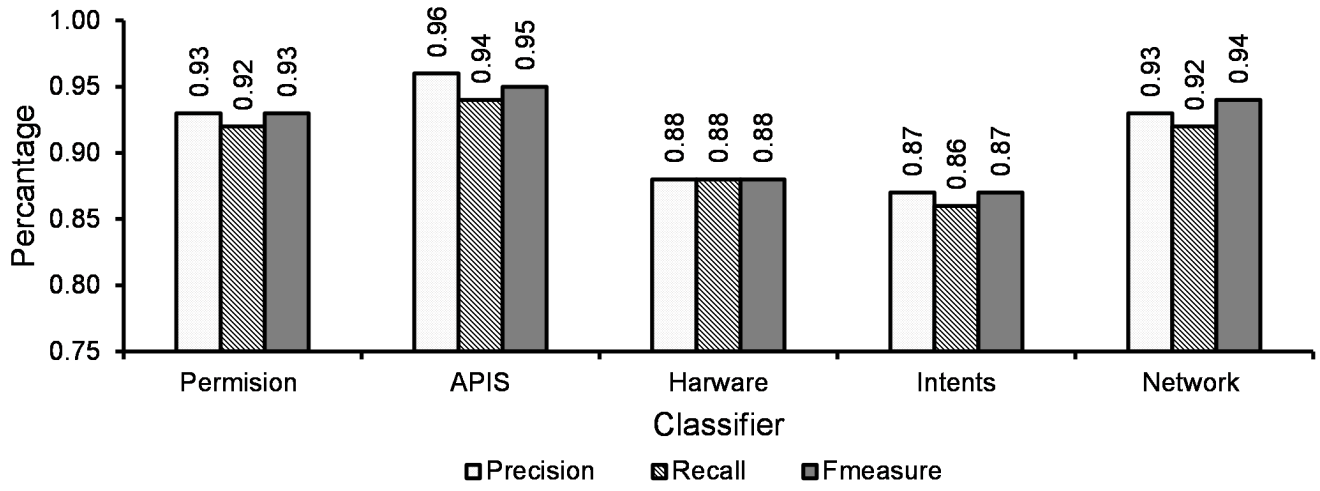

Fig. 2. Performance of propose model using TPOT Generation 1.

than that of the permission results shown in Fig. 3. Moreover, the average F-measure for hardware-based features class is 0.87 whereas, the average precision-recall is 0.87 as well. Similarly, Figs. 2 and 3 also present the classification results for intent-based features class. The average F-measure for the intent-based features class is 0.85 whereas, the precision-recall is $0.84 - 0.86$. Finally, Figs. 2 and 3 present the classification results for network addresses-based features. The average F-measure is 0.935 and the average precision-recall is $0.92 - 0.92$. We have used 3,888 malware samples out of $5,560$ total malware samples for classification in the network addresses-based features class. We were not able to find any network address in $1,672$ malicious samples from the Derbin dataset. However, for $3,888$ malicious samples containing network addresses, we achieve high accuracy and rank network addresses-based feature class at position five because of feature absence in 30.1% of malicious samples.

An overview of results for all the five subsets of features (APIs, Intents, Hardware Components, Permissions and Network Addresses) is presented in Table II. As shown in the table II, classifiers trained on these feature sets individually have the potential to classify malicious and benign apps individually. Hence, we select the top 4 discriminating feature subsets to train the proposed model. Although the fifth-ranked feature subset, i.e., Network addresses, has a high detection accuracy. We still do not select it to be a part of the proposed framework because the Network-based features are missing in 30.1% of malicious samples.

V. Countermeasure for Adversarial Attacks

In this section, we present a proposed framework to mitigate adversarial evasion attacks on ML-based classification models and prove the effectiveness of the proposed model in adversarial environments.

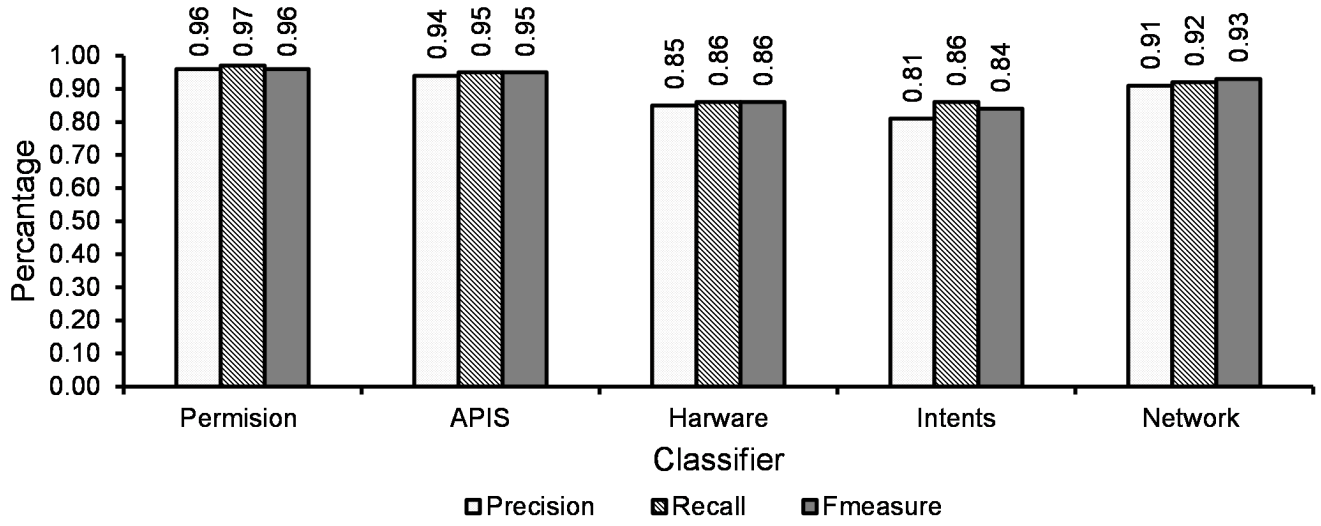

Fig. 3. Performance of propose model using TPOT Generation 10.

TABLE II
CLASSIFICATION RESULTS FOR FEATURE SUBSETS

	Precision	Recall	F-measure
Permissions	0.940	0.939	0.939
API	0.852	0.928	0.888
Hardware	0.857	0.798	0.827
Intents	0.745	0.924	0.825
Network	0.854	0.954	0.901

In general, there are three potential solutions to mitigate evasion attacks using ML: (1) training the target classifier with adversarial examples called adversarial training; (2) classifier ensembles; and (3) making target classifiers hard to attack. In this paper, we focus on options 2 and 3 by employing ensemble classifiers and making the target model hard to attack. Consequently, our proposed salable classification model can be employed to construct an adversarial evasion attacks resilient framework. As shown in Fig. 4, the learning model employs multiple classifiers where each classifier is trained on different subsets of features independently to generate a final output label. We use four top-ranked feature subsets (permissions, APIs, intents and hardware components) to design the proposed model. Each classifier in the pool is trained on these subsets separately to generate a label. Finally, the proposed model generates a conclusive label for a sample under observation by performing *OR* operation on the results from each classifier in the pool. Consequently, if an attacker fabricates some part of the application, e.g. APIs, the classifier trained on the APIs subset will fail. However, the proposed model will still detect the malicious App using the results from other classifiers in the pool trained on different subsets, e.g., permissions, intents or hardware-based features. Nonetheless, the proposed would still be vulnerable to evasion attacks. However, the process of evasion is made more complex for an attacker by employing our technique. As compared to traditional classifiers, e.g., Drebin [3], an attacker needs to make more changes in the original malicious sample to evade the proposed model. The model can be made even more complex to evade by incorporating more classifiers in the pool where each classifier is trained on different subsets of discriminating features. In the next section, we explain the effectiveness of the proposed model in adversarial settings by conducting an empirical case study.

As a proof of concept, we evade one of the states of the art classifiers in the domain of Android malware detection called Drebin [3]. The dataset used by Drebin is composed of $5,560$ malicious and $123,453$ benign apps. We have also used the same data set to rank the static features of an Android app to detect malware. Drebin extracts different features from Android apps, including hardware components, requested permissions, application components, filtered intents, local API calls, suspicious API calls, and used permissions and network addresses. Moreover, all these extracted features are embedded in a single multidimensional feature vector space. After the feature extraction process, Drebin uses linear *Support Vector Machines* (SVM). Drebin achieved a remarkable 94% recall on malware class with only 1% of FPR. To perform this case study, we replicated Drebin by providing the same data set. We used linear SVM to classify malicious and benign apps. The results of our experiments to replicate Drebin are presented in Fig. 4 by using a ROC curve plot. As shown in Fig. 4, an attacker can achieve 100% evasion of all the malicious samples in Drebin dataset by just fabrication of 3 samples without changing the semantics of the malicious entity. However, our findings suggest that the process of evasion can be made complex for the attacker using the proposed model. As explained previously, we found five subsets of most discriminating and meaningful features based on which we could classify most

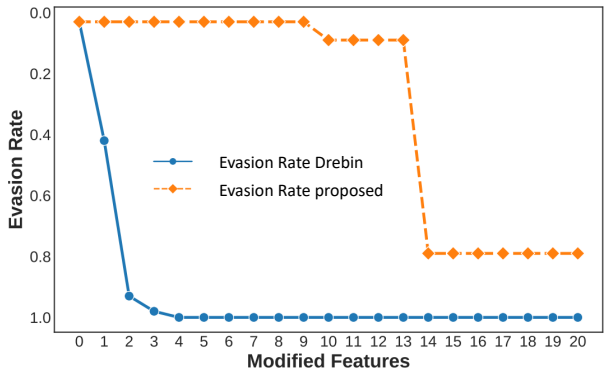

Fig. 4. Performance of proposed in adversarial environment.

malicious and benign samples separately. In this case study, we modified Drebin. Rather than training SVM on single vectors of combined features, we train it on four different feature sets (permissions, APIs, intents and hardware components) separately. Now, even though the attacker has access to the data and target classifier to extract the top features, the attacker will not evade the classifier easily. This is because the top features belong to a specific class of a subset. Hence changing one feature might affect the accuracy of one class (e.g.. Permissions). However, we still could detect the same malware with the help of other classes of subsets (e.g., APIs, intents and hardware components). As shown in Fig. 4, the same evasion attack as Drebin is applied to the proposed model. However, the proposed can correctly classify malware with 91% accuracy up to 14 changes in the original malicious feature vector. On the other hand, Drebin was 100% evaded with just three modifications in the malicious samples.

VI. Conclusion

In this study, we extract multiple subsets of discriminating features from the Android App for malware detection. Moreover, we performed an empirical case study to evade Drebin, a state-of-the-art Android malware classifier and propose a countermeasure to mitigate such attacks. Finally, our experiments show that Drebin can be 100% evaded by just fabricating three features in feature vector space.

Acknowledgment

This project is partially supported by the Western University of Applied Sciences, Bergen, Norway.

References

[1] Y. Aafer, W. Du, and H. Yin, "Droidapiminer: Mining api-level features for robust malware detection in android," in *International conference on security and privacy in communication systems*. Springer, 2013, pp. 86–103.

[2] A. Al-Dujaili, A. Huang, E. Hemberg, and U.-M. O'Reilly, "Adversarial deep learning for robust detection of binary encoded malware," in *IEEE Security and Privacy Workshops*. IEEE, 2018, pp. 76–82.

[3] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: Effective and explainable detection of android malware in your pocket." pp. 23–26, 2014.

[4] E. Aydogan and S. Sen, "Automatic generation of mobile malwares using genetic programming," in *European conference on the applications of evolutionary computation*. Springer, 2015, pp. 745–756.

[5] B. Biggio and F. Roli, "Wild patterns: Ten years after the rise of adversarial machine learning," *Pattern Recognit.*, vol. 84, pp. 317–331, 2018.

[6] A. Calleja, A. Martín, H. D. Menéndez, J. Tapiador, and D. Clark, "Picking on the family: Disrupting android malware triage by forcing misclassification," *Expert Systems with Applications*, vol. 95, pp. 113–126, 2018.

[7] A. Feizollah, N. B. Anuar, R. Salleh, G. Suarez-Tangil, and S. Furnell, "Androdialysis: Analysis of android intent effectiveness in malware detection," *computers & security*, vol. 65, pp. 121–134, 2017.

[8] J. Garcia, M. Hammad, and S. Malek, "Lightweight, obfuscation-resilient detection and family identification of android malware," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 26, no. 3, pp. 1–29, 2018.

[9] P. Laskov and R. Lippmann, "Machine learning in adversarial environments," *Mach. Learn.*, vol. 81, no. 2, pp. 115–119, 2010.

[10] J. Lee, S. Lee, and H. Lee, "Screening smartphone applications using malware family signatures," *computers & security*, vol. 52, pp. 234–249, 2015.

[11] K. Liu, S. Xu, G. Xu, M. Zhang, D. Sun, and H. Liu, "A review of android malware detection approaches based on machine learning," *IEEE Access*, vol. 8, pp. 124 579–124 607, 2020.

[12] N. Martins, J. M. Cruz, T. Cruz, and P. H. Abreu, "Adversarial machine learning applied to intrusion and malware scenarios: a systematic review," *IEEE Access*, vol. 8, pp. 35 403–35 419, 2020.

[13] G. Meng, Y. Xue, C. Mahinthan, A. Narayanan, Y. Liu, J. Zhang, and T. Chen, "Mystique: Evolving android malware for auditing anti-malware tools," in *Proceedings of the 11th ACM on Asia conference on computer and communications security*. ACM, 2016, pp. 365–376.

[14] R. S. Olson and J. H. Moore, "Tpot: A tree-based pipeline optimization tool for automating machine learning," in *Workshop on automatic machine learning*. PMLR, 2016, pp. 66–74.

[15] Y. Pan, X. Ge, C. Fang, and Y. Fan, "A systematic literature review of android malware detection using static analysis," *IEEE Access*, vol. 8, pp. 116 363–116 379, 2020.

[16] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[17] N. Peiravian and X. Zhu, "Machine learning for android malware detection using permission and api calls," in *2013 IEEE 25th international conference on tools with artificial intelligence*. IEEE, 2013, pp. 300–305.

[18] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, P. G. Bringas, and G. Álvarez, "Puma: Permission usage to detect malware in android," in *International Joint Conference CISIS'12-ICEUTE 12-SOCO 12 Special Sessions*. Springer, 2013, pp. 289–298.

[19] S. Sheen, R. Anitha, and V. Natarajan, "Android based malware detection using a multifeature collaborative decision fusion approach," *Neurocomputing*, vol. 151, pp. 905–912, 2015.

[20] K. Tam, A. Feizollah, N. B. Anuar, R. Salleh, and L. Cavallaro, "The evolution of android malware and android analysis techniques," *ACM Computing Surveys (CSUR)*, vol. 49, no. 4, pp. 1–41, 2017.

[21] W. E. Zhang, Q. Z. Sheng, A. Alhazmi, and C. Li, "Adversarial attacks on deep-learning models in natural language processing: A survey," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 11, no. 3, pp. 1–41, 2020.

[22] M. Zheng, P. P. Lee, and J. C. Lui, "Adam: an automatic and extensible platform to stress test android anti-virus systems," in *International conference on detection of intrusions and malware, and vulnerability assessment*. Springer, 2012, pp. 82–101.

[23] M. Zheng, M. Sun, and J. C. Lui, "Droid analytics: a signature based analytic system to collect, extract, analyze and associate android malware," in *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*. IEEE, 2013, pp. 163–171.