Android Malware Detection: Looking beyond Dalvik Bytecode

Tiezhu Sun, Nadia Daoudi, Kevin Allix, Tegawendé F. Bissyandé
SnT
University of Luxembourg
Luxembourg
Email: firstname.lastname@uni.lu

***Abstract*—Machine learning has been widely employed in the literature of malware detection because it is adapted to the need for scalability in vetting large scale samples of Android. Feature engineering has therefore been the key focus for research advances. Recently, a new research direction that builds on the momentum of Deep Learning for computer vision has produced promising results with image representations of Android bytecode. In this work, we postulate that other artifacts such as the binary (native) code and metadata/configuration files could be looked at to build more exhaustive representations of Android apps. We show that binary code and metadata files can also provide relevant information for Android malware detection, i.e., that they can allow to detect Malware that are not detected by models built only on bytecode. Furthermore, we investigate the potential benefits of combining all these artifacts into a unique representation with a strong signal for reasoning about maliciousness.**

I. Introduction

Android applications have pervaded virtually all aspects of the lives of hundreds of millions of consumers. Equipping over a billion devices, Android apps are at the center of various research initiatives, notably with respect to security. For example, malware detection research is a never-ending race where approaches quickly become obsolete as malware writers evolve their techniques to hide malicious payload inside different artifacts.

In the early days of Android, malicious behavior patterns were easily detected with simple static [1] and dynamic analysis [2] or based on similarity comparisons [3]. As malware evolved, researchers considered applying machine learning with a large focus on engineering various sets of features based on static or dynamic analysis outputs. While such statistical machine learning approaches [4]–[10] achieved good performance on literature benchmarks, the trend today is to investigate neural network architectures towards learning comprehensive representations of Android apps. Such representations exploit lexical information of bytecode [11] and abstract syntax tree representations of code [12], etc. More recently, inspired by the remarkable successes of Deep Learning in the field of computer vision, researchers have started to investigate Android malware detection methods based on image representations of apps [13], [14]. To advance research in this direction, Daoudi et al. [15] developed DexRay as a baseline approach for Android image-based malware detection. Our work is set in the same ambitious research agenda that DexRay laid down: how can we best exploit image representations of Android apps to detect malware?

A very significant reason for the success of deep learning in computer vision is that natural images contain rich semantic information which could provide meaningful features for recognition tasks. If the powerful feature learning capabilities of deep learning are to be used for the task of malware detection, representing apps as images is indeed an appealing prospect. Unfortunately, representing an APK (Android PacKage) as an image is not a straightforward endeavor: In particular, what should be captured in the image has not yet been defined nor studied. A model trained on images presenting a partial view of the objects under study will suffer from blind-spots in its recognition tasks.

Translated into the Android realm, an APK file is itself made of a collection of files, that contain artifacts of different nature. Among the various artifacts in an APK, DexRay and many prior approaches only consider the Dalvik bytecode (.dex files, that we will refer to as `.dex` throughout the paper). Several other approaches have instead extracted features from the Manifest XML file (that we will refer to as `.xml` throughout the paper), which contains metadata about an Android App. The native code (.so files *i.e.*, Shared Library files, that will be referring to as `.so`) is seldom considered as a signal source for malware detection. Yet, native code being challenging to analyze, it is a sweet spot to hide malicious behaviors.

In this paper, we seek to contribute to the systematization of knowledge around Android malware detection, by investigating the value of multiple types of artifacts in representing apps for malware detection. We make the following contributions:

- We evaluate the suitability of three major types of artifacts for Android malware detection;
- We assess whether these artifacts each bring added-value, or whether they are redundant;
- We investigate four possible methods of combining these artifacts into one Deep Learning approach.

More specifically, we investigate the following research questions:

- **RQ1:** Does each of the major artifacts in Android apps contain relevant information for Malware Detection?
- **RQ2:** How redundant is the information across three considered artifacts?

- **RQ3:** To what extent can the performance of Malware Detection be improved by combining these artifacts?

II. Experimental Setup

In this section, we first present a brief description of DexRay. Then, we present the dataset and the experimental setup used to conduct our experiments.

A. Background on DexRay

DexRay [15] is a recent work that presented and evaluated a simple image-based App representation for Android malware detection. It converts the Dalvik bytecode of Android apps into gray-scale vector image representations. Each byte in the `.dex` file is mapped to a pixel value in the generated image, and all images for apps are resized to one single size, independently of the size of the app or of its bytecode. The features extraction and the classification are both carried out automatically using a simple 1-dimensional convolutional neural network architecture. This approach has been proven to be highly effective in detecting Android malware by reporting an F1-score of 0.96.

B. Dataset

We reuse here the dataset that was used in DexRay [15][1], which was collected from AndroZoo [16]. The APKs (*i.e.*, Android PacKages) in the DexRay dataset are from the period of 2019 to 2020. Benign apps are defined as the apps that have not been detected by any antivirus from VirusTotal[2]. The malware collection contains the apps that have been detected by at least two antivirus engines. We present in Table I a summary of this dataset. Since not all the APKs contain native code, we also summarize in the same Table the number of apps that include `.so` files.

TABLE I: Summary of DexRay Dataset

	Nbr of APKs	Nbr of APKs with `.so` files
Benign apps	96 858	63 037
Malware apps	61 696	56 678
Total	158 554	119 715

C. Experimental Methodology

In our experiments, we adopt the same experimental setup of DexRay. Specifically, we use the Hold-out strategy [17] by dividing the dataset into 80% for training, 10% for validation and 10% for testing. The detection model is trained on the training dataset, and the model hyper-parameters are optimized based on its performance on the validation dataset. The above process is repeated 10 times by randomly shuffling and splitting the dataset.

We train each model for a maximum of 200 epochs, and we stop the training using an early stopping strategy (we set the patience step to 50). Regarding the convolutional layer, we set the kernel size to 12 and we use `relu` [18] as the activation function. The two convolutional layers contain 64 and 128 channels respectively. The model architecture also contains two dense layers with a `sigmoid` activation function [19]: a first layer with 64 neurons, and a second layer with one neuron for the classification.

As we have presented in Section II-B, some APKs do not contain native code (*i.e.*, `.so` files). Thus, we have set each pixel to zero in the corresponding gray-scale image (*i.e.*, blank image) when the `.so` files are missing in a given APK.

We evaluate the performance of our models using four standard metrics: Accuracy, Precision, Recall and F1 score.

III. Empirical Investigation

A. RQ1: Does each of the major artifacts in Android apps contain relevant information for Malware Detection?

To investigate this RQ, we train a model for each of the three major types of artifacts: `.dex`, `.xml`, and `.so` files. The `.dex` model is thus trained only on dex files (as in the original DexRay paper). For the `.so` model, we build the images by considering the bytes of the `.so` file(s) of an APK instead of the `.dex` file(s)[3]. Similarly, in the `.xml` model, the bytes of the Manifest file are used as a source for the images.

As recommended in the DexRay paper [15], we choose the size of $(1, 128 \times 128)$, as it is the best trade-off between the model accuracy and computational efficiency. We conduct our experiments using the experimental methodology presented in Section II-C and we present our results in Table II.

TABLE II: Detection performance for each type of artifacts on the DexRay dataset

Source	Accuracy	Precision	Recall	F1-score
`.dex`	0.972	0.974	0.953	0.963
`.so`	0.953	0.985	0.892	0.936
`.xml`	0.94	0.918	0.927	0.923

As shown in Table II, the detection models have achieved F1-scores of 0.963, 0.936, and 0.923. From the overall results, we can conclude that `.so` files and `.xml` files can indeed provide relevant information for detecting malware, and the detection performance is competitive. We note that the `.so` model obtains a Recall of 0.892, even though over 8% of the malware do not have any `.so` files.

RQ1 Answer: The major artifacts (*i.e.*, `.dex` files, `.so` files and `.xml` files) in Android apps contain relevant information for malware detection. Although the `.so` and the `.xml` models do not reach the same performance as the `.dex` model, they still perform relatively well.

B. RQ2: How redundant is the information across three considered artifacts?

The models that are trained on `.so` and `.xml` files have shown promising scores for malware detection, but have a lower Recall than the model built on `.dex`. Hence it is possible that the malware detected by the `.so` and `.xml` models are just a subset of the malware detected by the `.dex` model. An explanation for such a case could be that the `.so` and `.xml`

[1]https://github.com/Trustworthy-Software/DexRay

[2]https://www.virustotal.com/

[3]In case no `.so` file is present, a blank image is used.

artifacts do not bring additional information compared to the `.dex` artifacts. The goal of RQ2 is to investigate whether `.so` and `.xml` could provide complementary information to `.dex` models. Therefore, we examine the overlap between the samples that are detected by each of the three models, as well as the samples that are detected by a model and missed by another. We present our results in Table III.

TABLE III: Overlap and Differences of predictions made by `.dex`, `.so`, and `.xml` models

		Detected by			Missed by		
		`.dex`	`.so`	`.xml`	`.dex`	`.so`	`.xml`
Detected by	`.dex`	na	14 917	14 655	na	476	738
	`.so`	14 917	na	14 415	186	na	688
	`.xml`	14 655	14 415	na	242	482	na

As shown in Table III, most of the samples can be correctly classified (*i.e.*, malware or benign apps) by the three models. At the same time, there are malware samples that the `.dex` model fails to detect but the `.so` model and/or the `.xml` model detect and vice versa. Specifically, in all 15 855 APKs in the test dataset, `.so` model can detect 186 APKs that escape the detection of `.dex` model. Also, the `.xml` model successfully detects 242 APKs that the `.dex` model fails to detect. This observation suggests that there is knowledge that could be harnessed from the three different sources of information to enhance the detection performance of the `.dex` model.

RQ2 Answer: Most of the apps are detected by the three models, indicating that most of the information contained in the different artifacts is redundant in the context of malware detection. Nevertheless, each model detects malware that the other models do not. Therefore, it is expected that a model built on all three sources of artifacts could outperform a single-source model.

C. RQ3: To what extent can the performance of Malware Detection be improved by combining these artifacts?

As we have concluded in Section III-B, the three types of artifacts can complement each other. In this section, we investigate different methods to combine the information provided by the `.dex`, the `.so`, and the `.xml` files. In the following, we present a brief description of four malware detection approaches that consider information from the three types of artifacts.

1) Long-vector images: For this model, we generate one image that contains the concatenation of the gray-scale 1-d images horizontally, i.e., the resulting image has a $(1, 128 \times 128 \times 3)$ dimension. Since the Long-vector images have the same form of the gray-scale 1-d images (They only differ in the width), we rely on the same model architecture of DexRay.

2) Rectangular images: For the *rectangle images*, the three gray-scale 1-d images are stacked vertically, i.e., the resulting image has a $(3, 128 \times 128)$ dimension. The same model architecture of DexRay is used with the rectangular images.

3) Color images: As a potential method to combine the three sources of artifacts, we investigate a "color" image, where each color ("channel") of the resulting image is built from the gray-scale image for one artifact source. We use the same model architecture of DexRay, and we apply minor modifications in order to make it compatible with the 3-channel image. Specifically, we use MaxPooling2D with $pool\_size = (12, 1)$, instead of MaxPooling1D with $pool\_size = (12)$. As for the convolutional layers, we have not made any change since the Keras API[4] automatically adapts to inputs with different dimensions. In our case, the 1-d convolutional layer thus has 3 kernels for the 3-channel input image.

4) Ensemble Model: In the ensemble version (as shown in Figure 1), we separately learn the features from each gray-scale 1-d image. Similar to the model for gray-scale image, each type of image undergoes its corresponding model branch which consists of two 1-d convolutional layers and two 1-d pooling layers. Next, we use a Concatenation layer to combine the feature maps extracted from the three gray-scale images. Then, we add another 1-d convolutional layer to learn and combine features from the three images. For the classification, we use the same two dense layers of DexRay.

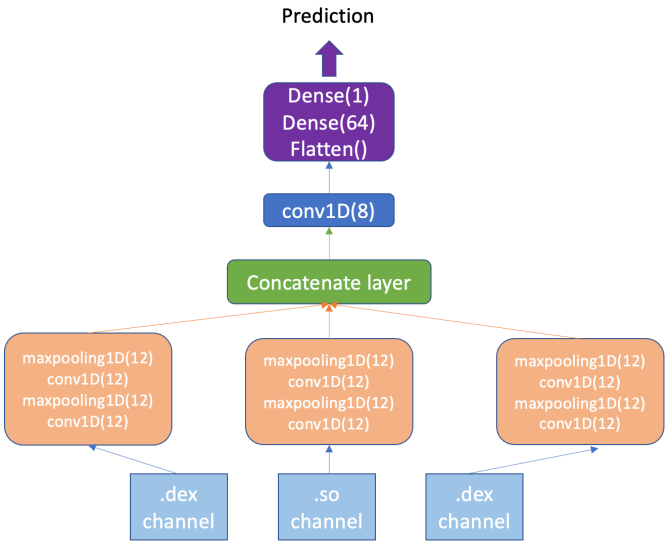

Fig. 1: Architecture of Ensemble Model for Color-scale Images

5) Results: We evaluate the detection performance of each of the previous models and we report our results in Table IV.

TABLE IV: Detection Results of combined sources on DexRay Dataset

Source	Accuracy	Precision	Recall	F1-score
long vector	0.969	0.97	0.949	0.959
rectangle	0.972	0.972	0.956	0.964
color	0.972	0.973	0.953	0.963
ensemble	0.974	0.973	0.959	0.966

Overall, we notice that the long-vector and the color image-based models do not improve the detection performance compared to the `.dex` image-based model. As for the rectangular

[4]https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv1D

and the ensemble models, they bring a slight improvement. In our experiments, the same dataset splits are used for the evaluation of all the model, which means that the ensemble model (i.e., the best combination method) outperforms the `.dex` model on the same exact applications. Although the ensemble model fails to detect 165 apps that are correctly predicted by the `.dex` model, it manages to detect 201 samples that have escaped the detection of the `.dex` model.

RQ3 Answer: Some combination methods can bring some improvements—although very minor—to the detection performance. None of the four proposed methods seem to fully benefit from the additional information brought by the different types of artifacts.

IV. Discussion

In this section, we first describe some insights about the image representations of Android apps for malware detection in Section IV-A. Then, we discuss some potential threats to validity of the proposed methods in Section IV-B. Finally, we discuss some future works which are worth studying next in Section IV-C.

A. Some Insights

In this work, we find that in addition to Dalvik bytecode, binary native code and Manifest files can also provide rich and relevant information for Android malware detection. Both the Dalvik bytecode and the Manifest files have been heavily used in the literature. However, to the best of our knowledge, there has been no comparative evaluation of their relative discriminating power in the context of malware detection.

Several ready-to-use tools have been developed to analyze `.dex` files. In contrast, analyzing Android apps' native code is not as streamlined a task. This could potentially explain why so few prior works leverage native code. One contribution of the present paper is to demonstrate that native code can indeed be leveraged without complex analysis, using a simple and straightforward representation.

An important point about Android Malware detection is that the performance exhibited by state-of-the-art approaches—and even by a single source `.dex` model—is already very high. Therefore, the potential gain in performance is very small, but nonetheless highly sought after: Even a mere 1% performance improvement could mean *thousands* of additional malware uncovered each month.

The present work offers a perspective to a simple yet important question: In the field of Android malware detection, where can improvements come from?

B. Threats to Validity

Our experiments and conclusions face some threats to validity. First, our experiments being conducted on a single dataset, the generalizability of our conclusions needs to be further verified. Nevertheless, the dataset we use is large, recent, and likely representative of what can be found on Android apps markets. It is furthermore made available to researchers for further experiments.

Second, the results we obtain may be specific to the image representation that we used. Indeed, how to best represent an Android app (or part of it) as an image is still an open question. By reusing the representation introduced in DexRay, our results are directly comparable to those of the original DexRay. Furthermore, we note that all the models we investigate in this paper achieve an f1-score above 0.9, suggesting that the representations used are adequate for Malware detection.

Similarly, the Deep-Learning architecture, and the training parameters we use may not be optimal. In particular, the combined artifacts may require a bigger model to be able to capture rich additional information.

C. Future Works

The present work established that there is information to be harnessed from the different types of artifacts. One open question is to investigate *how* to best leverage those various sources of information *together*. For example, here we have used the same size of image for each artifacts despite the significant difference in sizes of the artifacts: In an app, `.dex` files can amount to tens of Megabytes while the Manifest file is rarely more than tens of Kilobytes. Therefore an investigation of the optimal image size to use for each type of artifacts will help optimize a combination.

Another direction for improvement would be to investigate methods to concatenate/merge the representations of artifacts, since the four considered here that could not fully take advantage of all the artifacts.

We have evaluated the potential of three types of artifacts. Those do not represent the entirety of an Android app. Other artifacts could also be evaluated such as cryptographic certificates, images, sound files, User Interface layouts, or even external metadata like user reviews.

Finally, our work does not use different Deep-Learning architectures for the different artifacts. It is conceivable that the information of each artifacts could be best captured by a specific DL architecture.

V. Related Work

Android malware has been studied extensively in the literature. We introduce the fundamental technologies for Android analysis in Section V-A. Then, we present existing image representation methods of Android apps in Section V-B. Finally, we introduce ML-based/DL-based methods and multi-view techniques for malware detection in Section V-A.

A. Android Analysis Technologies

Static and dynamic analysis are two main methods to analyze Android applications. Static analysis examines the application based on information extracted statistically from its APK, and dynamic analysis studies the execution and behavior of the app when it is running. Many studies have investigated Android malware detection based on static analysis, which focus on information extracted from the artifacts of the APK: permissions [20], sensitive API calls [21], Dalvik Executable (DEX) file [22], control-flow and data-flow graphs [23], [24],

and specific control flow patterns that is in native code level [25].

As an effective supplement to static analysis, dynamic analysis also plays an important role in Android malware detection. It can help find dynamic malicious behavior [26], [27], collect data from physical devices [28], and provide visualized information [29].

B. Image Representation of Android Apps

Many learning-based methods rely on image representation of Android apps, which could be roughly divided as gray-scale images and color-scale images. A usual way to generate gray-scale images is to regard bytes of artifacts in APKs as pixel values [30]–[32], or to just convert the Manifest file to permission vectors [33]. Other works that use color image representations are generally based on the `.dex` files [13], [34], or on an analysis of `.dex` files [14], [35]. In both cases, despite using color images, those approaches rely on a single source of artifacts.

C. Malware Detection Methods

1) ML-based and DL-based Methods: Machine learning approaches [4]–[7], [36] usually extract hand-crafted features from the apps. After converting features to vectors, a ML classifier is used for the detection task.

In recent few years, many Android malware detection approaches [37]–[39] based on deep learning have been proposed. However, most of these methods rely on a variety of hand-crafted features, and some other methods leverage advanced model architectures such as ResNet [40], and Inception-v4 [41]. Recently, Daoudi *et al.* have proposed a baseline pipeline [15] for image-based malware detection with straightforward steps.

2) Multi-view Techniques: Most of the malware detection approaches mentioned above rely on a single source of artifacts for their features. In order to pursue higher and more robust performance, some researchers has considered multi-view techniques to incorporate different single-view features for malware detection. As being leveraged in previous studies [42]–[44], common choices for multi-view features are app permissions, API calls, sensor usage, and proprietary Android API package usage, *etc.*. Other approaches [45]–[47] have considered OpCodes, ByteCodes, header information, attacker's intent and system-calls multi-view features. In [48], [49], a unified framework that systematically integrates multiple views of the apps to perform comprehensive malware detection has been proposed.

VI. Conclusion

We demonstrate that the image representations of the three major types of artifacts considered in this work contain relevant information for malware detection. We show that each type of artifacts brings its very own valuable information, absent from the other artifacts. Indeed, the models that rely on the `.xml` and the `.so` images can detect android apps that escape the detection of the `.dex` model. We then experiment with four approaches to construct an image representation of Apps that encompass all three types of artifacts. Some of the tested approaches achieve slightly better performance.

Our work highlights the potential benefits of combining multiple sources of artifacts, and identifies the need for a strengthened research effort on combination methods, and on the evaluation of artifacts' value.

VII. Data Availability

All artifacts are available online at:
https://github.com/Trustworthy-Software/Looking-beyond-Dalvik-Bytecode

Acknowledgment

This work was partially supported by the Fonds National de la Recherche (FNR), Luxembourg, under project CHARACTERIZE C17/IS/11693861, and by the University of Luxembourg under the HitDroid grant.

References

[1] H. Kang, J.-w. Jang, A. Mohaisen, and H. K. Kim, "Detecting and classifying android malware using static analysis along with creator information," *International Journal of Distributed Sensor Networks*, vol. 11, no. 6, p. 479174, 2015.

[2] T. Petsas, G. Voyatzis, E. Athanasopoulos, M. Polychronakis, and S. Ioannidis, "Rage against the virtual machine: hindering dynamic analysis of android malware," in *Proceedings of the seventh european workshop on system security*, 2014, pp. 1–6.

[3] P. Faruki, V. Ganmoor, V. Laxmi, M. S. Gaur, and A. Bharmal, "Androsimilar: robust statistical feature signature for android malware detection," in *Proceedings of the 6th International Conference on Security of Information and Networks*, 2013, pp. 152–159.

[4] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: Effective and explainable detection of android malware in your pocket." in *Ndss*, vol. 14, 2014, pp. 23–26.

[5] 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.

[6] E. Mariconti, L. Onwuzurike, P. Andriotis, E. De Cristofaro, G. Ross, and G. Stringhini, "Mamadroid: Detecting android malware by building markov chains of behavioral models," *arXiv preprint arXiv:1612.04433*, 2016.

[7] H. Fereidooni, M. Conti, D. Yao, and A. Sperduti, "Anastasia: Android malware detection using static analysis of applications," in *2016 8th IFIP international conference on new technologies, mobility and security (NTMS)*. IEEE, 2016, pp. 1–5.

[8] H. Cai, N. Meng, B. Ryder, and D. Yao, "Droidcat: Effective android malware detection and categorization via app-level profiling," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 6, pp. 1455–1470, 2018.

[9] W.-C. Wu and S.-H. Hung, "Droiddolphin: a dynamic android malware detection framework using big data and machine learning," in *Proceedings of the 2014 Conference on Research in Adaptive and Convergent Systems*, 2014, pp. 247–252.

[10] F. Martinelli, F. Mercaldo, and A. Saracino, "Bridemaid: An hybrid tool for accurate detection of android malware," in *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, 2017, pp. 899–901.

[11] W. Niu, R. Cao, X. Zhang, K. Ding, K. Zhang, and T. Li, "Opcode-level function call graph based android malware classification using deep learning," *Sensors*, vol. 20, no. 13, 2020. [Online]. Available: https://www.mdpi.com/1424-8220/20/13/3645

[12] J. Zhang, X. Wang, H. Zhang, H. Sun, K. Wang, and X. Liu, "A novel neural source code representation based on abstract syntax tree," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, 2019, pp. 783–794.

[13] T. Hsien-De Huang and H.-Y. Kao, "R2-d2: Color-inspired convolutional neural network (cnn)-based android malware detections," in *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018, pp. 2633–2642.

[14] Y.-X. Ding, W.-G. Zhao, Z.-P. Wang, and L.-F. Wang, "Automaticlly learning featurs of android apps using cnn," in *2018 International Conference on Machine Learning and Cybernetics (ICMLC)*, vol. 1. IEEE, 2018, pp. 331–336.

[15] N. Daoudi, J. Samhi, A. K. Kabore, K. Allix, T. F. Bissyandé, and J. Klein, "Dexray: A simple, yet effective deep learning approach to android malware detection based on image representation of bytecode," in *Deployable Machine Learning for Security Defense*, G. Wang, A. Ciptadi, and A. Ahmadzadeh, Eds. Cham: Springer International Publishing, 2021, pp. 81–106.

[16] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "Androzoo: Collecting millions of android apps for the research community," in *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*. IEEE, 2016, pp. 468–471.

[17] S. Raschka, "Model evaluation, model selection, and algorithm selection in machine learning," *arXiv preprint arXiv:1811.12808*, 2018.

[18] A. F. Agarap, "Deep learning using rectified linear units (relu)," *arXiv preprint arXiv:1803.08375*, 2018.

[19] S. Sharma and S. Sharma, "Activation functions in neural networks," *Towards Data Science*, vol. 6, no. 12, pp. 310–316, 2017.

[20] F. Alswaina and K. Elleithy, "Android malware permission-based multi-class classification using extremely randomized trees," *IEEE Access*, vol. 6, pp. 76 217–76 227, 2018.

[21] O. Mirzaei, G. Suarez-Tangil, J. M. de Fuentes, J. Tapiador, and G. Stringhini, "Andrensemble: Leveraging api ensembles to characterize android malware families," in *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, 2019, pp. 307–314.

[22] Y. Fang, Y. Gao, F. Jing, and L. Zhang, "Android malware familial classification based on dex file section features," *IEEE Access*, vol. 8, pp. 10 614–10 627, 2020.

[23] X. Zhiwu, K. Ren, and F. Song, "Android malware family classification and characterization using cfg and dfg," in *2019 International Symposium on Theoretical Aspects of Software Engineering (TASE)*. IEEE, 2019, pp. 49–56.

[24] Z. Xu, K. Ren, S. Qin, and F. Craciun, "Cdgdroid: Android malware detection based on deep learning using cfg and dfg," in *International Conference on Formal Engineering Methods*. Springer, 2018, pp. 177–193.

[25] S. Alam, Z. Qu, R. Riley, Y. Chen, and V. Rastogi, "Droidnative: Automating and optimizing detection of android native code malware variants," *Computers & Security*, vol. 65, pp. 230–246, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016740481630164X

[26] S. K. Dash, G. Suarez-Tangil, S. Khan, K. Tam, M. Ahmadi, J. Kinder, and L. Cavallaro, "Droidscribe: Classifying android malware based on runtime behavior," in *2016 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2016, pp. 252–261.

[27] A. Martín, V. Rodríguez-Fernández, and D. Camacho, "Candyman: Classifying android malware families by modelling dynamic traces with markov chains," *Engineering Applications of Artificial Intelligence*, vol. 74, pp. 121–133, 2018.

[28] L. Massarelli, L. Aniello, C. Ciccotelli, L. Querzoni, D. Ucci, and R. Baldoni, "Android malware family classification based on resource consumption over time," in *2017 12th International Conference on Malicious and Unwanted Software (MALWARE)*. IEEE, 2017, pp. 31–38.

[29] S.-W. Hsiao, Y. S. Sun, and M. C. Chen, "Behavior grouping of android malware family," in *2016 IEEE International Conference on Communications (ICC)*. IEEE, 2016, pp. 1–6.

[30] F. Mercaldo and A. Santone, "Deep learning for image-based mobile malware detection," *Journal of Computer Virology and Hacking Techniques*, pp. 1–15, 2020.

[31] J. Jung, J. Choi, S.-j. Cho, S. Han, M. Park, and Y. Hwang, "Android malware detection using convolutional neural networks and data section images," in *Proceedings of the 2018 Conference on Research in Adaptive and Convergent Systems*, 2018, pp. 149–153.

[32] K. Bakour and H. M. Ünver, "Deepvisdroid: android malware detection by hybridizing image-based features with deep learning techniques," *Neural Computing and Applications*, pp. 1–18, 2021.

[33] M. Ganesh, P. Pednekar, P. Prabhuswamy, D. S. Nair, Y. Park, and H. Jeon, "Cnn-based android malware detection," in *2017 International Conference on Software Security and Assurance (ICSSA)*. IEEE, 2017, pp. 60–65.

[34] X. Xiao, "An image-inspired and cnn-based android malware detection approach," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 1259–1261.

[35] G. D'Angelo, M. Ficco, and F. Palmieri, "Malware detection in mobile environments based on autoencoders and api-images," *Journal of Parallel and Distributed Computing*, vol. 137, pp. 26–33, 2020.

[36] J. Shawe-Taylor and N. Cristianini, *An introduction to support vector machines and other kernel-based learning methods*. Volume, 2000, vol. 204.

[37] E. B. Karbab, M. Debbabi, A. Derhab, and D. Mouheb, "Maldozer: Automatic framework for android malware detection using deep learning," *Digital Investigation*, vol. 24, pp. S48–S59, 2018.

[38] M. K. Alzaylaee, S. Y. Yerima, and S. Sezer, "Dl-droid: Deep learning based android malware detection using real devices," *Computers & Security*, vol. 89, p. 101663, 2020.

[39] W. Wang, M. Zhao, and J. Wang, "Effective android malware detection with a hybrid model based on deep autoencoder and convolutional neural network," *Journal of Ambient Intelligence and Humanized Computing*, vol. 10, no. 8, pp. 3035–3043, 2019.

[40] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[41] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Thirty-first AAAI conference on artificial intelligence*, 2017.

[42] A. Appice, G. Andresini, and D. Malerba, "Clustering-aided multi-view classification: a case study on android malware detection," *Journal of Intelligent Information Systems*, vol. 55, no. 1, pp. 1–26, 2020.

[43] S. Singh, K. Chaturvedy, and B. Mishra, "Multi-view learning for repackaged malware detection," in *The 16th International Conference on Availability, Reliability and Security*, 2021, pp. 1–9.

[44] S. Millar, N. McLaughlin, J. M. del Rincon, and P. Miller, "Multi-view deep learning for zero-day android malware detection," *Journal of Information Security and Applications*, vol. 58, p. 102718, 2021.

[45] H. Darabian, A. Dehghantanha, S. Hashemi, M. Taheri, A. Azmoodeh, S. Homayoun, K.-K. R. Choo, and R. M. Parizi, "A multiview learning method for malware threat hunting: windows, iot and android as case studies," *World Wide Web*, vol. 23, no. 2, pp. 1241–1260, 2020.

[46] S. M. Hazrati Fard and E. Velayati, "Malware detection and identification using multi-view learning based on sparse representation," *International Journal of Web Research*, vol. 2, no. 2, pp. 45–53, 2019.

[47] J. Bai and J. Wang, "Improving malware detection using multi-view ensemble learning," *Security and Communication Networks*, vol. 9, no. 17, pp. 4227–4241, 2016.

[48] A. Narayanan, M. Chandramohan, L. Chen, and Y. Liu, "A multi-view context-aware approach to android malware detection and malicious code localization," *Empirical Software Engineering*, vol. 23, no. 3, pp. 1222–1274, 2018.

[49] A. Kyadige, E. M. Rudd, and K. Berlin, "Learning from context: A multi-view deep learning architecture for malware detection," in *2020 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2020, pp. 1–7.