Towards Multi-view Android Malware Detection Through Image-based Deep Learning

Jhonatan Geremias*, Eduardo K. Viegas†*, Altair O. Santin*, Alceu Britto*, Pedro Horchulhack*
*Pontificia Universidade Catolica do Parana (PUCPR) — Graduate Program in Computer Science (PPGIa), Brazil
{jhonatan, santin, alceu, pedro.horchulhack}@ppgia.pucpr.br
†Secure Systems Research Center — Technology Innovation Institute (TII), Abu Dhabi
eduardo@ssrc.tii.ae

2022 International Wireless Communications and Mobile Computing (IWCMC) | 978-1-6654-6749-0/22/$31.00 ©2022 IEEE | DOI: 10.1109/IWCMC55113.2022.9824985

***Abstract*—Over the last years, several works have proposed highly accurate Android malware detection techniques. Surprisingly, modern malware apps can still pave their way to official markets, thus, demanding the provision of more robust and accurate detection approaches. This paper proposes a new multi-view Android malware detection through image-based deep learning, implemented threefold. First, apps are evaluated according to several feature sets in a multi-view setting, thus, increasing the information provided for the classification task. Second, extracted feature sets are converted to an image format while maintaining the principal components of the data distribution, keeping the information for the classification task. Third, built images are jointly represented in a single shot, each in a predefined image channel, enabling the application of deep learning architectures. Experiments on a new version of a publicly available Android malware dataset composed of over 11 thousand Android apps have shown our proposal's feasibility. It reaches true-negative rates of up to 99.5% when implemented with a single-view approach with our new image-building technique. In addition, if our proposed multi-view scheme is used, the classification accuracies of malware families become more stable, reaching a true-positive rate of up to 98.7%.**

***Index Terms*—Android Malware Detection, Deep Learning, Static Analysis.**

I. Introduction

Nowadays, it is estimated that over 2.5 billion of active devices are making use of the Android mobile operating system, accounting for over 70% of the total smartphones' market share [1]. The number of malicious Android applications, namely *malwares*, are also on the rise, currently affecting 24% of all Android users [2]. An indication that current techniques used for securing Android users have failed considering that over 67% of all *malware* apps were originated from official app markets [2].

As a result, over the last years, several techniques have been proposed for Android *malware* detection, which can be typically divided into two main approaches [3]. On the one hand, *dynamic-based* techniques evaluate the behavior of the Android application at execution time, in general, making use of a sandbox environment, posing several challenges related to the generation of the proper app stimulus for the triggering of the malicious app behavior [4]. Notwithstanding, modern *malwares* is even able to detect when they are being monitored in a sandbox setting, thus, remaining undetectable by such techniques [5]. On the other hand, *static-based* approaches evaluates the app characteristics in an offline manner, typically through the contents of the *Android Application Pack* (*apk*) file, such as the requested app permissions (*manifest*), native compiled codes (*lib*), or even the Java compiled source files (*dex*) [6]. As a result, it significantly eases the detection process, as it does not requires executing the analyzed Android app sample.

Therefore, several *static-based* detection techniques have been proposed over the last years, wherein a recent and promising approach relies on the application of deep learning architectures [7]. The authors first convert the analyzed file (e.g., *dex*) into an image format, typically through a direct image translation in a byte-to-pixel approach, where the built image is resized according to the input size of the used deep learning algorithm [8]. Consequently, both the byte-to-pixel and the image resizing processes can introduce significant tradeoffs in the image classification task, considering that meaningful feature values are often lost.

Notwithstanding, current deep learning techniques for Android malware detection often rely on a single file for the image conversion task [9]. For instance, focusing on providing a high detection accuracy through a single *apk* file as input, such as the *dex*, *lib*, or *manifest* files. The behavior of Android *malwares* can often only be identified when analyzed through a variety of perspectives (views) [10]. For instance, an analyzed app may request several permissions to be granted (*manifest*), a characteristic that is often identified in *malwares*. However, if the source code (*dex*) is also analyzed, such granted permissions may be identified as correctly used by the app. Current approaches in the literature are often able to provide high detection accuracies in a specific test set. However, they fail to reach similar accuracies when a more complex dataset is considered [11]. This is because a single view is often unable to depict the characteristics of all Android app samples in production. This property is only achieved through the evaluation of several feature sets.

This paper proposes a new multi-view Android malware detection through image-based deep learning, implemented threefold. First, to-be-classified Android apps are evaluated considering a multi-view setting, thus, assessing several feature sets extracted through the corresponding app *apk* content. Consequently, the analyzed apps can be classified according to various views, increasing the detection generalization and

reliability. Second, extracted feature sets are converted to an image format by identifying the principal components of the data distribution. Therefore, the feature sets are mapped to an image taking into account their feature value distribution, thus, maintaining meaningful features relation during such a process. Third, built view images are grouped in a single shot, each in a corresponding image channel. The main insight of the proposed model is to be able to apply deep learning techniques while maintaining the original Android app feature distribution in a multi-view setting.

In summary, the main paper contributions are:

- An evaluation of current single view Android malware detection techniques. Such an evaluation shows that classification accuracy is highly related to the used feature set and varies according to the test dataset.
- A new multi-view Android malware detection through image-based deep learning. The evaluation shows that our proposed model can provide similar accuracy to the most accurate single-view techniques.

The remainder of this paper is organized as follows. Section II further describes the application of ML techniques on Android malware detection. Section III describes related works. Section IV presents our proposal, while Section V evaluates its performance. Section VI concludes our work.

II. Preliminaries

This section further describes the typical Android *malware* detection challenge, including the modules that implement such a task. Then, we further discuss the challenges related to the application of deep learning classification to perform the Android *malware* detection.

A. Android Malware Detection

Static-based Android malware detection tasks can often be represented through four sequential modules [12]. First, the *Data Acquisition* module receives as input the data to be used for the classification task, which is often performed through the Android Application Package (*apk*) file. Second, the *Feature Extraction* module compounds a feature set according to the used feature view. For instance, analyzing the native compiled source codes (*lib*) or the Java compiled source codes (*dex*). The built feature vector is then used as input to a *Classification* module, which applies a machine learning (ML) model to classify its input as either *normal* or *malware* class [13]. Analyzed *malware*-classified samples are signaled by the *Alert* module.

Over the last years, several techniques have been proposed for the classification task in Android malware detection, wherein authors often resort to ML-based approaches implemented as a pattern recognition task [3]. A behavioral ML model is built according to the training dataset data composed of huge amounts of both *normal* and *malware* Android samples [14]. The built ML model can then be used in production for the classification of additional Android *malware* samples. Surprisingly, despite the high accuracy rates reported in the literature, traditional ML-based approaches have been unable to secure users against *malware* apps, as noted by the frequent reports of malicious apps paving their way to the official app markets [2]. As a result, several works in the literature have been pursuing new malicious app detection approaches.

B. Deep Learning Android Malware Detection

Techniques based on deep learning architectures are currently being considered the state-of-the-art in image recognition and classification tasks, being successfully used in several fields, such as medical diagnosis, fraud detection, and object recognition [15], [16]. As a result, several works have proposed the application of deep learning techniques for Android *malware* detection [17]. However, deep learning was proposed initially for image-based problems, making its application for Android *malware* detection a challenging task.

To enable the application of deep learning techniques, Android *malware* detection schemes must address two main challenges, *image representation* and *image resizing* [17]. The *image representation* concerns the translation of the analyzed file from the Android *apk* to an image format. In their vast majority, proposed approaches execute such a task in a direct byte-to-pixel value translation, thus, representing the vector into a matrix format. The built image size varies according to the used *apk* file, e.g., a *dex* file size may change according to the Android app size, from a few megabytes to up to several gigabytes, thus, resulting in a variable-sized image. Therefore, the *image resizing* task aims to standardize the generated image size, a requirement for the application of traditional deep learning techniques. In general, current approaches perform the image resizing without considering the original distribution of the feature values. As a result, the image used as input by the deep learning architecture may not properly depict the analyzed app characteristics. It may lose such properties during the image resizing task.

III. Related Works

Over the last years, several highly accurate Android *malware* detection approaches have been proposed in the literature, typically through ML-based schemes. For instance, J. Jiang *et al.* [18] analyzes the Android app opcode sequence calls for the building of an ML feature vector. Their proposed model provided reasonably high accuracies but overlooked the application of deep learning and the evaluation of additional feature sets. In contrast, K. Zhao *et al.* [19] proposes an ML-based *malware* detection model according to the app requested permissions and API calls. The proposed model increased accuracy by using traditional ML classifiers when both views were considered. Unfortunately, deep learning application was not taken into account. Similarly, a deep learning approach was proposed by D. Zhu *et al.* [20] to analyze the app data flows. The authors were able to increase accuracy compared to traditional ML classifiers. However, they neglected the multi-view detection and image-based deep learning application challenges.

In recent years, several works have been proposing the application of image-based deep learning architectures for Android *malware* detection [17]. For instance, D. Vasan. *et al.* [21] performs the detection of Android *malware* through the analysis of the apk *dex* file. To achieve such a goal, the authors convert the *dex* file into a byte-to-pixel approach and resize the generated image without taking into account the feature value distribution. Despite that, their proposed model increased accuracy while overlooking the evaluation of additional feature sets. P. Yadav *et al.* [22] proposes a new deep learning architecture to increase the *malware* detection accuracy. Their proposed model evaluates the apk *dex* file to address *malware* obfuscation detection. Unfortunately, the authors perform the traditional image resizing, neglecting the feature distribution and additional feature sets.

A more robust approach to address the image building process for deep learning applications was proposed by A. Darwaish *et al.* [8]. The authors rely upon a dictionary-based process for the translation of the apk *dex* file to an image format, showing that the careful evaluation of the useful features can improve detection accuracy. Similarly, the authors overlook the application of multi-view. J. Jung *et al.* [9] address the dimensional image challenge through the image size fixing according to the *dex* original file size. The authors can increase accuracy when fixing the original image size while overlooking the feature value distribution and the multi-view application. Therefore, it is possible to note that works in the literature often do not address the application of several feature sets in a multi-view setting. Nonetheless, deep learning-based schemes neglect the challenges related to the image-building task.

IV. A Multi-View Deep Learning Android Malware Detection Model

We propose a multi-view deep learning Android *malware* detection model aiming for higher detection accuracies through the evaluation of additional feature sets. Our proposed model addresses multi-view deep learning in a threefold manner and is shown in Figure 1.

First, to enable the implementation easiness of our proposed model, we extract several feature sets in a vector format. Our model uses the extracted feature sets (views) to build a corresponding image to apply image-based deep learning architectures. Second, to convert the extracted feature set to a proper image format, we use an image builder module that considers the underlying distribution of the feature values during such a conversion. The central insight of such a technique is to decrease the information losses caused by traditional image conversion and resizing approaches, thus, increasing the final model accuracy built over such data. Third, to properly enable the multi-view nature of our proposed scheme, we use the image joiner approach, wherein the built grayscale images are joined together to a single colored image. More specifically, each view, represented by a grayscale image, is depicted as a specific image channel, e.g., RGB channels. Consequently, the multi-view nature of the Android *malware* detection is adequately considered during the detection task without information losses caused by the image generation and image joiner modules.

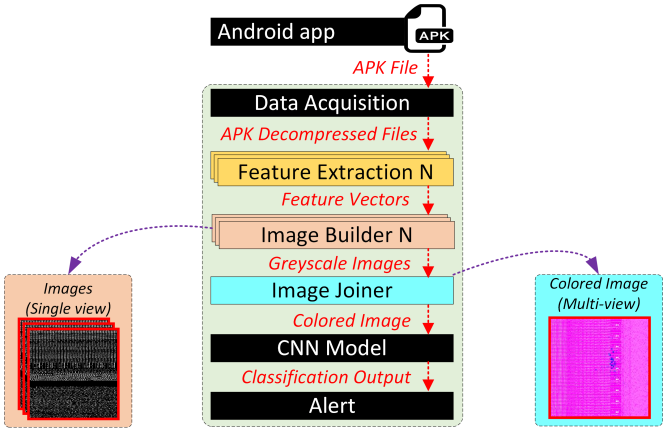


Fig. 1: Proposed multi-view deep learning Android malware detection model.

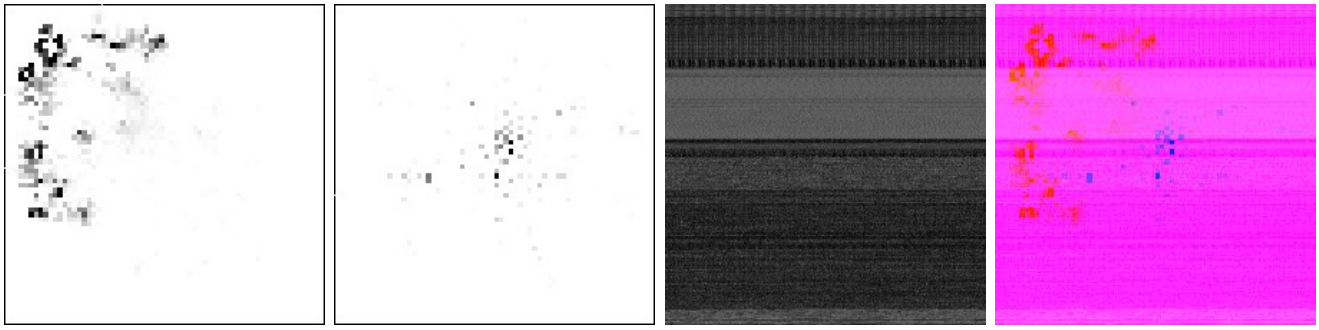
The following subsection describes our proposed model and the modules that implement it.

A. Image Building in a Multi-view Setting

Traditional approaches used for building the image that will be used as input by the deep learning model in their vast majority overlook the information losses caused by the image building and resizing tasks. Notwithstanding, the application of additional feature sets in a multi-view setting is overlooked. As a result, traditional approaches used for the image-building task may introduce information losses that will affect the final accuracy of the used deep learning algorithm.

To address such a shortcoming, our proposed model takes into account the original feature distribution values during the image building procedure (Fig. 1, *Image Builder N*). Similarly, as made through traditional approaches used to extract the principal components of the data distribution (e.g., Principal Component Analysis (PCA)), we convert each feature vector to a matrix format by applying a feature transformation scheme. More specifically, for each given N-sized input feature vector, we output a MxM-sized matrix, computed through the extraction of the principal components of the data distribution. As a result, the built image can keep the original feature distribution, despite the image building and resizing tasks. In addition, to enable the proper evaluation of all built images, wherein each is made according to each extracted feature set, we create a new image that depicts the multi-view setting of our model. A new colored image with N channels is built to achieve such a goal, wherein each channel represents each extracted view image. Therefore, a single image is generated for the classification task that appropriately depicts each considered view by our model.

The following subsection further describes the classification pipeline used by our model that uses our image-building procedure.



(a) API Calls (b) OPCodes (c) Dex (d) Multi-view

Fig. 2: Sample *malware* images built through our proposed feature reduction for image builder technique. The built images are used as input according to the used view for each selected deep learning algorithm.

B. Image-based Classification

The classification procedure, shown in Figure 1, starts with a to-be-classified Android *apk* file. The ingested *apk* file is decompressed and the *apk* files properly selected. For instance, the *dex*, *manifest.xml*, and *lib* files. The selected files are provided as input to a set of feature extraction modules; wherein each module outputs a corresponding feature vector (Fig. 1, *Feature Extraction N*). The extracted feature vectors are converted into a set of grayscale images by the *Image Builder* module, which maintains the original feature distribution during such a process (see Section IV-A). The built images are provided to the *Image Joiner* module, which goal is to make a single image by allocating each generated grayscale image into a specific image channel. Thus, a single image is generated in a colored format, wherein each channel represents the original view. The built colored image is used as input by a deep learning model for the classification task, providing the classification output to an alert module, which signals *malware*-classified apps to the user.

C. Discussion

Our proposed model aims to enable the application of multi-view Android *malware* detection through image-based deep learning architectures. To achieve such a goal, we perform the image building task through a feature transformation technique, which can maintain the original feature distribution during such a task. As a result, the image conversion can be performed without the image resizing approach, significantly improving the information that is provided to the subsequent modules. To enable the application in a multi-view setting, we build a single image composed of several channels, wherein each channel represents a specific view. Consequently, our proposed scheme can perform the image-based deep learning application in a multi-view setting without degrading the information available during the classification task.

V. Evaluation

The evaluation of our proposed model aims at answering three main research questions:

- (***RQ1***) *How does deep learning techniques perform when using our proposed image builder technique?*
- (***RQ2***) *How our proposed multi-view model perform for Android malware detection?*
- (***RQ3***) *How does our proposed model perform when compared to traditional approaches?*

The following subsections show the obtained results, including the model building procedure and the evaluation results.

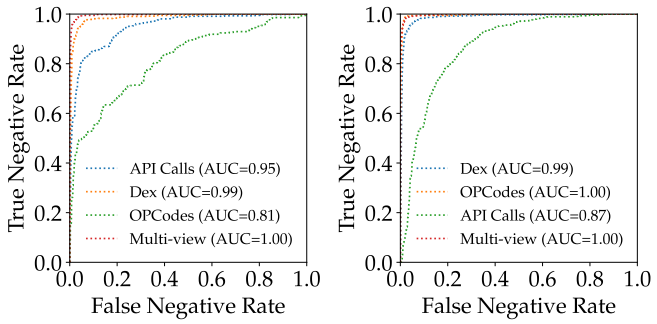
A. Multi-view Android Malware Dataset

Current approaches for Android *malware* detection in the literature often make use of a single feature set for detection purposes. Consequently, the detection task is bound to a single view (feature set), e.g., an API call view can only provide detection features for apps that change their behavior at the API call level.

To address such a shortcoming, we enhance a publicly available Android *malware* dataset to include additional feature sets. To achieve such a goal, we explore the widely used *CICMalDroid* [23] dataset, made of $11,598$ Android app samples, those distributed in $1,795$ *normal* Android apps, and 4 additional *malware* families, composed by $1,253$ *adware*, $2,100$ *banking*, $3,904$ *smsfraud*, and $2,546$ *riskware* apps. For each Android app sample from the dataset, we extract 3 views (feature set) as follows:

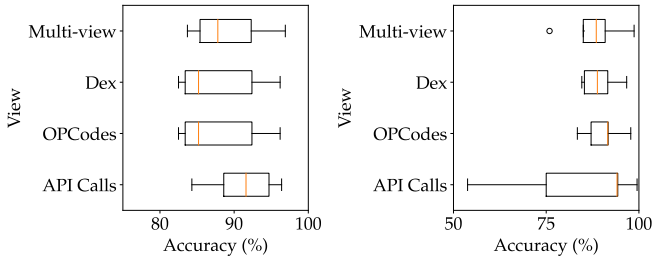
- ***API Calls***. $2,426$ features represent the number of calls a given Android app made to each function in a predefined list.
- ***OPCodes***. 216 features representing the OPCode occurrence, from a predefined list of OPCodes, in the Android *dex* file.
- ***Dex***. $50,176$ features obtained through a traditional byte-to-pixel translation, according to the Android *dex* file.

To extract the aforementioned listed features, we make use of the AndropyTool [24], which analyzes the Android *apk* file and outputs a corresponding *json* file. The produced file is then post-processed to standardize its format across the $11,598$ analyzed Android app samples



(a) InceptionV3 (b) Resnet50

Fig. 3: Receiver operating characteristics (ROC) curve for each selected deep learning architecture according to the used feature set (view) over the test dataset.



(a) InceptionV3 (b) Resnet50

Fig. 4: Accuracy distribution for each evaluated view and deep learning architecture. The proposed model was able to provide the highest accuracies regardless of the used deep learning architecture.

B. Model Building

Our model was implemented by making use of two widely used deep learning architectures, namely *InceptionV3* and *Resnet50*. Each architecture was evaluated executing for $1,000$ epochs, and its learning rate was set empirically according to the resulting loss and a momentum weight of 0.9. The architectures were implemented through $keras$ API $v.2.4.0$, and $tensorflow$ API $v.2.4.1$. The proposed image building procedure (see Section IV-A) was implemented through the DeepInsight [25] API $v.2$ while making use of the *t-Distributed Stochastic Neighbor Embedding* feature reduction technique. An additional view was used to evaluate our technique to make use of the multi-view approach when building the 3 image color channels. To achieve such a goal, we build a traditional *dex* image in a byte-to-pixel format. Figure 2 shows an example of the built images through our proposed image building technique.

The classifiers were evaluated according to their True-Positive (TP) and True-Negative (TN) rates. The TP denotes the ratio of *malware* instances correctly classified as *malware*, while the TN denotes the ratio of *normal* instances correctly classified as *normal*. The built dataset was randomly split in *training*, *testing*, and *validation* datasets, each composed by 40%, 30% and 30% of Android apps respectively. A random undersampling without replacement is used in the *training* dataset to balance the occurrence between the classes.

C. Multi-view Android Malware Detection

The first experiment aims at answering *RQ1* and evaluates the classification performance of the selected deep learning architectures when implemented by making use of a single view. To achieve such a goal, each selected architecture is trained over the *training* dataset, while using the *validation* dataset for the generalization evaluation, and using the *testing* dataset to report the obtained accuracies (see Section V-A for dataset split). Figures 3a, and 3b shows the ROC curve for the single view deep learning architectures. It is possible to note that the single-view approaches provided significantly high detection accuracies using our proposed image-building technique. For instance, the selected single-view techniques that were implemented through the InceptionV3 architecture, were able to provide an Area-Under-the-Curve (AUC) values of 0.95, 0.81 and 1.00 for the *API Calls*, *OPCodes*, and *Dex* views respectively. Table I further investigates the classification accuracies of the single-view approaches. Similarly, the accuracy rates were significantly high, for instance, the Resnet50 deep learning model presented a true-negative rate of 99.5%, 91.7%, and 91.6% for the *API Calls*, *OPCodes*, and *Dex* views respectively. As a result, our proposed image-building scheme enables the application of deep learning techniques without losing information caused by the image building and image resizing techniques.

TABLE I: Classification accuracy of selected techniques according to the used feature set (view) over the validation dataset.

View	Deep Learning	Classification Accuracy (%)				
		Norm.	Ad.	Bank.	Risk.	SMS
API Calls	Inception	91.6	88.6	84.3	94.7	96.4
	Resnet50	99.5	94.3	75.0	53.8	94.1
Opcode	Inception	85.2	92.4	82.5	83.4	96.2
	Resnet50	91.7	83.4	87.1	91.6	97.8
Dex	Inception	73.6	73.2	76.0	87.9	95.9
	Resnet50	91.6	88.8	84.6	85.3	96.7
Multi-View	Inception	92.3	85.4	83.7	87.8	96.9
	Resnet50	90.9	85.0	75.9	88.5	98.7

Our second experiment aims at answering *RQ2* and evaluates our proposed model while making use of the multi-view image (Fig. 2d). The selected deep learning architectures were built with the multi-view generated image, in a colored format, with each image channel representing a single view (see Fig. 2d, for a sample image). Figure 3 shows the ROC curve for the multi-view approaches for each selected deep learning architecture, while Table I shows the individual accuracies according to each *malware* family. It is possible to note that the multi-view deep learning architectures could improve detection accuracy compared to single-view ones, regardless of the used architecture. For instance, the AUC of the multi-view technique was 1.0 for both architectures, thus, providing similar accuracies to the best practices based on single-view.

The proposed model, which uses a multi-view setting, could portray the specificity of every single view in a single image, as can be noted by the provision of the highest accuracy regardless of the used deep learning architecture.

Finally, to answer *RQ3*, we further investigate the classification improvements obtained by our proposed model when compared to traditional techniques. Table I shows the classification accuracies of our proposed model when making use of single and multi-view settings. It is possible to note that our proposed multi-view scheme, which relies on our image building technique, can provide the most stable accuracies compared to the single-view ones. Figure 4 shows the distribution of the *malware* classification accuracies according to the used view. Our proposed multi-view approach provided the most stable classification values compared to the single-view ones in such a case. Consequently, the proposed scheme can provide higher accuracies for Android *malware* detection while considering additional feature sets for the classification task.

VI. Conclusion

Authors from the literature propose approaches for detecting Android *malwares*, but despite their extensive efforts, these threats are still on the rise. This paper has addressed Android *malware* detection through image-based deep learning techniques. The proposed model considers the image building and resizing challenges through feature reduction techniques, which can keep the principal components of the feature distribution during the image building task. The experiments performed in a widely used Android *malware* dataset have shown our proposal feasibility, increasing the accuracy when compared to both single-view and traditional approaches. In future works, we plan on extending the evaluation for other platforms *malwares* and also evaluate different views.

Acknowledgment

This work was partially sponsored by Brazilian National Council for Scientific and Technological Development (CNPq) grants 430972/2018-0 and 306684/2018-7.

References

[1] T. inMobi, "Understanding android users worldwide," 2021. [Online]. Available: https://www.inmobi.com/blog/2021/08/09/understanding-android-users-worldwide

[2] P. Kotzias, J. Caballero, and L. Bilge, "How did that get in my phone? unwanted app distribution on android devices," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, May 2021, p. 53–69.

[3] J. Qiu, J. Zhang, W. Luo, L. Pan, S. Nepal, and Y. Xiang, "A survey of android malware detection with deep neural models," *ACM Computing Surveys (CSUR)*, vol. 53, no. 6, pp. 1–36, Feb. 2021.

[4] F. H. da Costa, I. Medeiros, T. Menezes, J. V. da Silva, I. L. da Silva, R. Bonifácio, K. Narasimhan, and M. Ribeiro, "Exploring the use of static and dynamic analysis to improve the performance of the mining sandbox approach for android malware identification," *Journal of Systems and Software*, vol. 183, p. 111092, Jan. 2022.

[5] T. Vidas and N. Christin, "Evading android runtime analysis via sandbox detection," in *Proceedings of the 9th ACM symposium on Information, computer and communications security*. ACM, Jun. 2014.

[6] R. Taheri, M. Ghahramani, R. Javidan, M. Shojafar, Z. Pooranian, and M. Conti, "Similarity-based android malware detection using hamming distance of static binary features," *Future Generation Computer Systems*, vol. 105, pp. 230–247, Apr. 2020.

[7] R. R. dos Santos, E. K. Viegas, and A. O. Santin, "A reminiscent intrusion detection model based on deep autoencoders and transfer learning," in *2021 IEEE Global Communications Conference (GLOBECOM)*. IEEE, Dec. 2021. [Online]. Available: https://doi.org/10.1109/globecom46510.2021.9685724

[8] A. Darwaish and F. Nait-Abdesselam, "RGB-based android malware detection and classification using convolutional neural network," in *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*. IEEE, Dec. 2020.

[9] J. Jung, J. Choi, S. je 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*. ACM, Oct. 2018.

[10] 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, May 2021.

[11] D. Arp, E. Quiring, F. Pendlebury, A. Warnecke, F. Pierazzi, C. Wressnegger, L. Cavallaro, and K. Rieck, "Dos and don'ts of machine learning in computer security," in *31st USENIX Security Symposium*, 2022.

[12] P. Horchulhack, E. K. Viegas, and A. O. Santin, "Toward feasible machine learning model updates in network-based intrusion detection," *Computer Networks*, vol. 202, p. 108618, Jan. 2022.

[13] B. B. Bulle, A. O. Santin, E. K. Viegas, and R. R. dos Santos, "A host-based intrusion detection model based on OS diversity for SCADA," in *IECON 2020 The 46th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, Oct. 2020.

[14] F. Ramos, E. Viegas, A. Santin, P. Horchulhack, R. R. dos Santos, and A. Espindola, "A machine learning model for detection of docker-based APP overbooking on kubernetes," in *ICC 2021 - IEEE International Conference on Communications*. IEEE, Jun. 2021.

[15] E. Viegas, A. O. Santin, and V. A. Jr, "Machine learning intrusion detection in big data era: A multi-objective approach for longer model lifespans," *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 1, pp. 366–376, Jan. 2021.

[16] J. Mallmann, A. O. Santin, E. K. Viegas, R. R. dos Santos, and J. Geremias, "PPCensor: Architecture for real-time pornography detection in video streaming," *Future Generation Computer Systems*, vol. 112, pp. 945–955, Nov. 2020.

[17] Z. Wang, Q. Liu, and Y. Chi, "Review of android malware detection based on deep learning," *IEEE Access*, vol. 8, pp. 102–126, 2020.

[18] J. Jiang, S. Li, M. Yu, G. Li, C. Liu, K. Chen, H. Liu, and W. Huang, "Android malware family classification based on sensitive opcode sequence," in *2019 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, Jun. 2019.

[19] K. Zhao, D. Zhang, X. Su, and W. Li, "Fest: A feature extraction and selection tool for android malware detection," in *2015 IEEE Symposium on Computers and Communication (ISCC)*. IEEE, Jul. 2015.

[20] D. Zhu, H. Jin, Y. Yang, D. Wu, and W. Chen, "DeepFlow: Deep learning-based malware detection by mining android application for abnormal usage of sensitive data," in *2017 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, Jul. 2017.

[21] D. Vasan, M. Alazab, S. Wassan, H. Naeem, B. Safaei, and Q. Zheng, "IMCFN: Image-based malware classification using fine-tuned convolutional neural network architecture," *Computer Networks*, vol. 171, p. 107138, Apr. 2020.

[22] P. Yadav, N. Menon, V. Ravi, S. Vishvanathan, and T. D. Pham, "EfficientNet convolutional neural networks-based android malware detection," *Computers & Security*, vol. 115, p. 102622, Apr. 2022.

[23] S. Mahdavifar, A. F. A. Kadir, R. Fatemi, D. Alhadidi, and A. A. Ghorbani, "Dynamic android malware category classification using semi-supervised deep learning," in *2020 IEEE Intl Conf on Dep., Aut. and Secure Computing (DASC)*. IEEE, Aug. 2020, pp. 515–522.

[24] A. Martín, R. Lara-Cabrera, and D. Camacho, "Android malware detection through hybrid features fusion and ensemble classifiers: The AndroPyTool framework and the OmniDroid dataset," *Information Fusion*, vol. 52, pp. 128–142, Dec. 2019.

[25] A. Sharma, E. Vans, D. Shigemizu, K. A. Boroevich, and T. Tsunoda, "DeepInsight: A methodology to transform a non-image data to an image for convolution neural network architecture," *Scientific Reports*, vol. 9, no. 1, Aug. 2019.