An Intelligent Malware Detection and Classification System Using Apps-to-Images Transformations and Convolutional Neural Networks

Farid Naït-Abdesselam[†]*, Asim Darwaish* and Chafiq Titouna*
*University of Paris, France
[†]University of Missouri Kansas City, USA

***Abstract*—With the proliferation of Mobile Internet, handheld devices are facing continuous threats from apps that contain malicious intents. These malicious apps, or malware, have the capability of dynamically changing their intended code as they spread. Moreover, the diversity and volume of their variants severely undermine the effectiveness of traditional defenses, which typically use signature-based techniques, and make them unable to detect the previously unknown malware. However, the variants of malware families share typical behavioral patterns reflecting their origin and purpose. The behavioral patterns, obtained either statically or dynamically, can be exploited to detect and classify unknown malware into their known families using machine learning techniques. In this paper, we propose a new approach for detecting and analyzing a malware. Mainly focused on android apps, our approach adopts the two following steps: (1) performs a transformation of an APK file into a lightweight RGB image using a predefined dictionary and intelligent mapping, and (2) trains a convolutional neural network on the obtained images for the purpose of signature detection and malware family classification. The results obtained using the Androzoo dataset show that our system classifies both legacy and new malware apps with high accuracy, low false-negative rate (FNR), and low false-positive rate (FPR).**

Keywords: Android malware, Deep learning, Classification.

I. Introduction

The proliferation of handheld devices, endowed with numerous sensors (such as barometer, gyroscope, and positioning sensors), has unlocked a new era of smart-apps development for end-users, contributing in the generation of highly sensitive information, which need paramount attention to protect from malicious activities. Therefore, it is imperative to establish a robust defense line against malware apps for handheld devices. The detection and isolation of malware is challenging due to the nature, diversity, obfuscation, and other sophisticated ways of malicious code. Moreover, with the exponential growth of mobile apps, it is notably very challenging to examine each application manually for malicious behaviors.

Malware detection techniques follow mainly two approaches: (1) a static analysis, where an analysis is done before the execution of the malware, or (2) a dynamic analysis, where an analysis is done during the execution of the malware. Following a static analysis approach, we propose in this paper a novel malware detection system which transforms first any Android APK file into an RGB image, independently from feature engineering and source code analysis. Using the resulted RGB images from transformation, we train a convolutional neural network (CNN) for signature detection and classification of different malware in their respective families. To set a combat force against an android malware challenge, we aim to build a lightweight, robust, and scalable malware detector without using feature engineering techniques, parameter configuration and zero code analysis. The main contributions of this research work are:

Novel Transformation of an APK file to an RGB Image: Following the principle of static analysis, we opted for a reverse engineering tool, called *Androguard* [1], to analyze the android application packaging file and collect all the permissions, activities, services, receivers, providers, and their intents, as well as strategically map the android manifest.xml to the green channel of an image. Secondly, we collect and map all the API calls and opcode sequences into the red channel of an image. Lastly, we devise the exhaustive predefined dictionary [2] for suspected API calls, permissions, activities, services, and receivers to map them to a blue channel, along with protected strings without redundant information. We have used interpolation techniques to keep the image size consistent for training the classifier.

Scalable and robust solution: The proposed system is scalable and capable of detecting malware ranges from 2009 to date. The system can also detect evolving, varying, sophisticated, and polymorphic malware apps without source code analysis and fingerprint features. A lightweight reverse engineering tool Andaguard expedites the static analysis of large APK samples. We have evaluated the performance of the system on AndroZoo repository over various balance datasets. In addition to signatures detection, the system also performs image-based malware family classification. We have evaluated our approach for the top 10 malware families provided in the AndroZoo dataset.

The rest of the paper is organized as follows: Section II provides the related research work. Section III articulates our proposed system for APK transformation and android malware detection. Section IV contains the experimental setup and performance of the system. Finally, section V summarizes the contributions and opens future directions.

II. Related Work

This section circumvents the related research work for android malware detection using machine learning and deep learning approaches following static or dynamic analysis track.

Malware apps are continuously emerging, and new variants are being introduced daily, traditional approaches like signature-based and handcrafted feature selection have limitations to cope with this challenge and struggles to detect new malware and their corresponding families. Rule-based and various machine learning techniques have been in practice for a decade to detect both static [3]–[6] and dynamic [7], [8] malware detection. These static approaches heavily rely on handcrafted feature selection, such as intents, API calls, permissions, etc. using machine learning techniques, for instance, Naive-bayesian and K-nearest neighbor [9], Support Vector Machine (SVM) [3]. Moreover, n-gram based malware detection systems also used to distinguish between benign and malware by incorporating low-level opcodes [10]. A traditional ML approach (SVM) is used for malware detection [4], which uses permissions mining to determine the most contributing and significant permission for the classification of malware. Their goal was to use the minimum number of permissions to analyze the malicious behavior of an app and ended up with 22 relevant permissions. Their accuracy is not very high as compared to other deep learning-based approaches and nor identifying the malware family.

With the advancement of deep learning, Neural Networks are widely acceptable in Computer Vision and Natural Language Processing domains. Similarly, [11] used the Recurrent Neural Network (RNN) for malware classification using the hybrid approach, including static and dynamic features. An alternative of n-gram based malware detection approach, [12], used a convolutional neural network to process the raw Dalvik bytecode and detect malware by capitalizing the raw opcode sequence by eliminating the feature engineering. However, this approach is blind in terms of static and dynamic analysis and did not explain the type of malware and have no clue about apps permissions. Another Neural network-based approach [13] has been proposed for malware detection via sequences mining. This approach uses the raw sequences of API calls that appear in disassembled code (DEX file). The purposed approach has limitations for code obfuscation and ignoring other critical contributing features.

Existing malware techniques are not robust to detect new malware families and struggling on the automatic extraction of fine-grained features. We have designed a system that is independent of feature engineering technique, generalized, lightweight, scalable, and converts an APK file to RGB image by intelligently mapping the dex file and manifest file on a different channel for malware detection and family classification.

III. Proposed Approach

This section explains our proposed approach for RGB-based malware detection. The section starts with APK file structure, conversion of APK file components to different channels, normalization of obtained images, and training of the convolutional neural network. The source code of complete system is available at [2].

A. *Transformation of Android Application Packaging (APK) file*

Java is the language choice for all android applications and compiled into Dalvik bytecode wrapped in a .dex file. APK is the Zip format used by Android OS for the distribution and installation of android application on any smartphone. The main structure of APK is mentioned as:

- META-INF/: manifest file, exhibit Java jar file information
- lib/: compiled code in .so format from NDK (Android Native Kit)
- res/: non compiled resources.
- AndroidManifest.xml/: posses configuration file for access control, authorization, version, and services.
- classes.dex/: Android exe file, java code compiled in Dalivk bytedcode.

Our approach starts with the AndroidManifest.xml file and collects all the permissions and app components (activities, services, receivers, providers, and their intents) using the reverse engineering tool *Androguard*. Androguard is lightweight and has fast extraction time as compared to NinjaDroid [14] and other tools. We convert all declared permission and app component into pixel values and place them on the green channel, excluding suspected permissions and app components found in predefined directory (exhaustive dictionary of suspected API calls, permissions, and app components formulated based on previous studies). The conversion is straightforward; we take the ASCI code of each character from permissions and app components. We arrange them in a specific order to generate a pattern in the resultant green channel, as depicted in Figure 1.

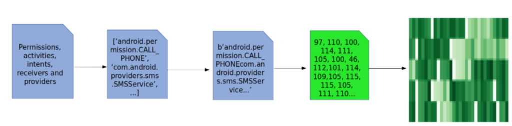

Fig. 1: Green Channel: Conversion of Permissions and app components from AndroidManifest.XML

After the manifest file, the system preforms the static analysis of a dex file and extract all API calls and opcode sequences from Dalvik bytecode. Excluding suspected API calls, we convert all other API calls to a decimal number by summing every ASCII character represented as (C) and take the modulus by 255 in order to normalize in a pixel range [0-255] as per equation 1, where C represents the ASCI character value in each API call.

$$API_{PixelValue} = (\sum_{i=1}^{n} C_i) \bmod 256 \tag{1}$$

For instance, Landroid/telephony/TelephonyManager-getDeviceId is and API call and conversion is given as under: Decimal values = [76, 97, 110, 100, 114, 111, 105, 100, 47, 116, 101, 108, 101, 112, 104, 111, 110, 121, 47, 84, 101, 108, 101, 112, 104, 111, 110, 121, 77, 97, 110, 97, 103, 101, 114, 59, 45, 62, 103, 101, 116, 68, 101, 118, 105, 99, 101, 73, 100]
Sum = 4793 Pixel-value = 4793 mod 255 = 203

Besides API mapping, there are millions of opcodes (operation code) in the dex file used in combination with different class-methods to execute various instructions at low level and few specific opcode sequences are used in malware. We have only used unique opcode sequences from the dex file and map on red channel. Because using all opcode makes the image noisy and can make the classification process cumbersome. Every opcode has a predefined value range from [0-255]. For example: invoke-super parameter, method-to-call: Invokes the virtual method of the immediate parent class. The opcode 'invoke-super' has hexa value of '6F' and the decimal value of '111'. Mapping of red channel can be visualized from the Figure 2

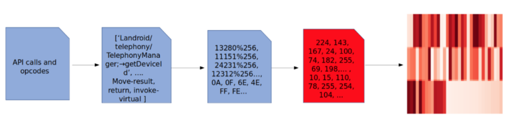

Fig. 2: Red Channel: Conversion of API calls and unique opcode sequences from Dex file

Finally, we collect all the suspected malicious permission and app components from the manifest file, suspected API calls, opcodes from dex, and protected strings and map their predefined pixel value on the blue channel, as shown in Figure 3. The system measures the similarity index of suspected malicious behavior with an exhaustive predefined dictionary formulated based on different malware studies [9], [15]–[17]. Moreover, if any suspected API call or permission is not listed in the exhaustive predefined dictionary, it is still being incorporated in the red and green channels and participates in the classification process. It is essential to highlight that Androguard does not only return protected strings which help in classification but also return rough strings which lead to noisy image. Therefore, we are only interested in malicious properties encapsulated in protected strings, which may be API call, permission, or any app component.

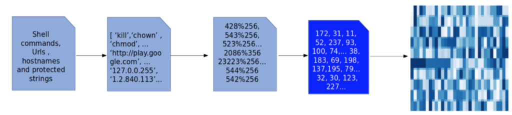

Fig. 3: Blue Channel: Conversion of protected strings,suspected permissions, app components and API calls

B. Consistent Image Size

Different Android applications have different APK size, the resultant image may have inconsistent channel size based on dex and manifest file. We need an RGB image with consistent channel size to feed our CNN classifier, we fix the image into 64x64 (height x width). If the size of the channel is less/greater than the specified threshold, we elongate or shrink the channel size by using interlinear interpolation. The complete transformation process of converting the APK into RGB image is shown in Algorithm 1.

Algorithm 1 Transformation of Android Application Packaging (APK) file to an RGB Image

1: **Predefined_Dictionary** suspicious properties from literature with assigned pixel values
2: **Begin**
3: $a, d, dx \Leftarrow$ Androguard(path/to/sha256.apk)
4: $green_channel,\ manifest_{susp} \Leftarrow$manifest2Pixel(a)
5: $Interpolation(green_channel,$size$= (64 \times 64))$
6: Convert Dex2Pixel(d, dx)
7: $red_channel, api_calls_{susp} \Leftarrow$api_calls2Pixel($dx$)
8: $Interpolation(red_channel,$size$= (64 \times 64))$
9: $opcodes_pixel \Leftarrow$opcodes2Pixel(dx)
10: $suspicious_strings \Leftarrow$strings2Pixel(d)
11: $blue\_channel \Leftarrow manifest_{susp} + api\_calls_{susp} + suspicious\_strings + opcodes\_pixel$
12: $Interpolation(blue_channel,$size$= (64 \times 64))$
13: $RGB \Leftarrow merge(red\_channel, green\_channel, blue\_channel)$
14: **End**

C. Convolutional Neural Network for Image-based Android Malware detection

Due to the inherent properties of images, Convolutional Neural Networks are widely used for images and continuous data. As now APKs are transformed into images and applying the CNN seems a smart choice for signature detection and family classification. The architecture of CNN for malware detection is depicted in Figure 4

From our experience and literature review, it is well known that sigmoid and hyperbolic tangent (tanh) results in gradient vanishing in back-propagation. Therefore, as depicted in figure 4 we have used rectified linear unit (ReLu) as an activation function. we uses 'sigmoid' activation in the output layer and for signature detection we use

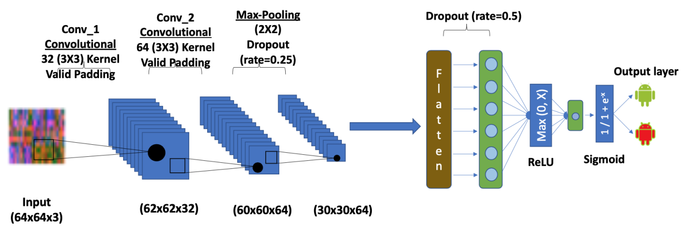

Fig. 4: Complete architecture of Deep Malware Detection System

'binary cross entropy' as a loss function given in equation 2 where, y stands for actual true label and $\hat{y}$ represents the predicted label. We have used the adaptive learning rate, Adam, with an initial learning rate set to 0.001, which decayed to 10^{-5} after each epoch. The rationale behind using Adam is its popularity for training the deep learning model with fast speed, plus it enables the power of Adagrad (Adaptive Gradient Algorithm) and RMSprop (Root Mean Square Propagation).

$$L(y,\hat{y}) = -\frac{1}{N}\sum_{i=0}^{N}(y * \log(\hat{y_i}) + (1-y) * \log(1-(\hat{y_i}))) \quad (2)$$

Figure 4 states that appetite to our system is the converted APK into RGB image with three channels and a size of 64 x 64. We start with a kernel size of 3 x 3 and filter number 32 at the first convolution layer. We use architecture consisting of two convolutional layers with Relu activation function followed by max pooling (2,2) and dropout with rate 0.25. After flattening the output of the dropout passes it through another dropout layer with a rate 0.5. The dropout layer is followed by a dense layer with glorot uniform and the output layer with sigmoid activation as depicted in Figure 4. We initialize the kernel with glorat also called xavier uniform [18] which draws samples from uniform distribution within the limit, where limit is defined as per equation 3.

$$limit = \sqrt{\frac{6}{fan_{in} + fan_{out}}} \quad (3)$$

where fan_{in} and fan_{out} shows the number of input and output in weight tensor respectively.

IV. Experiments and Discussion

This section describes the details of dataset, experimental setup, segregation of data, selection of hyper-parameters and results followed by discussion and future steps.

A. Details of Dataset

We are using AndroZoo [19] for evaluating our approach because it is a continuous growing repository and has more than 10 million android applications. These applications are labeled as malware and benign with multiple Anti Virus (AV) Engines. A total of 7.3 million applications marked as benign by gauging their Total Virus detection (VT) = 0. For our evaluation, we only choose malware apps; where VT detection is $\geq$ 15 and select apps from 2009 to 2020. AndroZoo provide SHA256 to access applications along with creation date which we use to segregate different datasets. The details of the dataset are given in Table I

TABLE I: Yearly view of Malware Samples in AndroZoo Dataset

Year	#APKs	VT ≥15	Year	#APKs	VT ≥15
2020	9978	3	2014	1762714	87400
2019	181825	1562	2013	772918	47925
2018	428725	5153	2012	566549	139169
2017	383406	5754	2011	219627	36900
2016	1425425	22781	2010	61122	1235
2015	866759	49280	2009	12075	61

B. Results for Signature Detection & Family Classification

We have segregated the datasets based on year and VT detection (Virus detection). For the first experiment, we have used the VT detection $\geq$ 35 and randomly collected 4K Malware and 4K Benign APKs from [2009-2011] using the SHA256 file provided by AndroZoo [19]. Similarly, the second experiment comprises of a dataset belongs to [2012-2014] and contains 12k malware and benign samples with VT detection $\geq$ 35. The third experiment, dataset pertinent to year range [2015-2016] with VT detection $\geq$ 15 and comprises 30K samples for both malware and benign. The final experiment contains data from the year range[2017-2020] with VT detection $\geq$15 and contains 11k latest malware and benign samples. As per our approach discussed in section III performs the novel transformation of APKs to RGB images. On top of these transformed images, we have trained a CNN and achieve remarkable results, as depicted in table II.

Malware family classification is important for security experts and anti-virus vendors to determine the threat level and establishing an apt defense mechanism. It is crucial for them to know about malware families and behaviors to educate and/or alert end users, how a particular malware can affect his/her mobile device. In AndroZoo dataset, there are total of 3305 different malware families and labels are not available from 2017-20 , for simplicity we have listed top 10 families and their counts in Table III.

We ran a series of experiments with different malware samples and VT detection for Android malware family classification. Using the SHA256 file, we have downloaded the top 10 malware families for simplicity, and the number of samples for each family is more than 5000. To the best of our knowledge, we are the first to use a huge number of family-wise samples for malware family classification. Following the same methodology discussed in section III, we have trained convolutional neural network (CNN) for malware samples having family labels available in AndroZoo dataset. The results are given in Table IV

TABLE II: RGB-based Malware Signature Detection Using CNN and ResNet

# M/B	VT Det	Year	FNR%		FPR%		F-1		AOC		Acc%	
			CNN	ResNet	CNN	ResNet	CNN	ResNet	CNN	ResNet	CNN	ResNet
4K Benign	0	[2009-2011]	1.7	0.62	1.5	0.25	0.98	0.99	0.997	1.0	98.21	99.41
4K Malware	≥ 35											
12K Benign	0	[2012-2014]	1.2	1.39	1.9	1.48	0.98	0.984	0.997	0.99	98.39	98.18
12K Malware	≥ 35											
30K Benign	0	[2015-2016]	4.9	2.7	3.5	2.3	0.95	0.96	0.977	0.977	95.73	97.42
30K Malware	≥ 15											
11K Benign	0	[2017-2020]	1.72	**0.85**	0.72	**0.39**	0.98	0.99	0.999	**1.0**	98.77	**99.37**
11K Malware	$\geq$15											

TABLE III: Top 10 Malware Families in AndroZoo Dataset

Rank	**Family**	**Count**	Rank	**Family**	**Count**
1	dowgin	262057	6	artemis	38041
2	kuguo	107114	7	droidkungfu	37336
3	airpush	100471	8	leadbolt	31491
4	revmob	74419	9	adwo	28733
5	youmi	51762	10	jiagu	27526

TABLE IV: Results of CNN for Malware Family Classification

# Families	**#Samples**	**VT Det**	**FPR%**	**F1**	**Acc%**
6	600/family	$\geq$35	9.7	0.91	91.21
10	175/family	$\geq$35	5.8	0.96	96
5	4100/family	$\geq$15	10.8	0.92	92.3
6	2300/family	$\geq$15	11.2	0.92	92.4
10	4100/family	$\geq$15	15.84	0.88	88.91

C. Discussion

The proposed approach is independent of feature engineering and extensive source code analysis. Moreover, it is very lightweight in terms of signature detection and family classification. The reason for being lightweight is the size of an APK file, which mostly varies between 4MB to 80 MB. As per the blog [20] an average app size for an android mobile application is 15MB and it varies for different mobile app categories. For example, an average size of a game app is 68MB and a navigation app is 45MB. Our on-device solution quickly converts an APK file into an RGB image of size 2-3KB in less than a second based on APK file size. Uploading an image of 3KB on the cloud for CNN based Malware detector and family classification is way cheaper and faster as compared to sending the whole or partial APK file. It is also imperative to mention that our approach is robust, practical, lightweight, and extremely suitable for android devices to detect legacy and newly-crafted malware compared to state-of-the-art approaches. Table V circumvents the comparison with a few latest and state of the art malware detection techniques.

Numerous other approaches have been proposed in the literature for android malware detection similarly to few given in the table V. Our approach is tested and validated on the largest malware dataset for both legacy and new malware without any feature engineering, as depicted in Table II and also performs the family classification of detected malware. [17], [21]–[23] used a total of 60K, 22K (2K malware only) , less than 4K malware, and 5.5k malware apps respectively for their experiments. Aforesaid approaches do not perform family classification; therefore, we have compared our result to a few other state of the art approaches for family classification. Comparison of Table IV and Table VI clearly articulates that earlier approaches are using old and very few malware samples for a particular class.

During experiments we have applied both CNN and ResNet (Residual Network) a flavour of Convolutional Neural Network for image recognition. We have not seen a substantial difference in results for malware signature detection, as depicted in Table II. We have used three different kinds of interpolation to normalize the resultant image size of an APK. These interpolation methods are Inter-area interpolation, inter-cubic interpolation and interlinear interpolation for generating pixel for new positions. Inter linear interpolation yielded better results as compared to other techniques due to the nature of the resultant image while other techniques are recommended for general objects within a scene.

TABLE V: Comparison of Malware Detection: Feature Engineering (FE) and Family Classification (FC)

Ref.	**FE**	**FC**	**Dataset**	**Acc%**	**FN%**
[17]	Yes	No	AMD+Drebin	97	3.16
[21]	Yes	No	Google.P+Genome	89.03	-
[24]	No	No	Google Play	93	9.0
[22]	Yes	No	Genome+Drebin	97.2	-
[25]	Yes	No	AndroZoo	97.65	-
[23]	Yes	No	Drebin	98	7.0
Our	**No**	**Yes**	AndroZoo	**98.77**	**1.7**

V. Conclusion

As Android is an open-source and malware samples continue to grow, this research work aims to build a deep learning-based malware detection system which can work for both legacy and new malwares. We have analyzed the android apps from 2009 to 2020 from the AndroZoo data repository without any feature engineering and source code analysis. The proposed approach performs the intelligent transformation of an APK file to an RGB image by mapping manifest file and dex file to corresponding channels. A CNN based malware classifier is trained and yielded excellent results as compared to previous approaches. Moreover, it is a lightweight and scalable solution for android devices. In the future, we are targeting our system to expose to different adversarial attacks to evaluate its robustness under different adversarial settings.

TABLE VI: Malware Family Classification Comparison

Ref.	# Samples	# Families	F-1	Dataset
[26]	59-833	15	0.82-0.95	Drebin
[27]	43-925	14	90.67	Drebin
[28]	43-925	20	0.90	Drebin
[29]	2408(total)	20	0.71	AMD
[30]	643(total)	43	91.1	FalDroid-II

References

[1] "Reverse engineering malware and goodware analysis of android applications, 6, 2017, [online] available: https://code.google.com/ archive/p/androguard/.," Last visited February 2020.

[2] "A. darwaish: Source code and dictionay, rgb-based-andorid-malware-detection [online] available: https://github.com/asimswati553/.," Last visited June 2020.

[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.," in *Ndss*, vol. 14, pp. 23–26, 2014.

[4] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-an, and H. Ye, "Significant permission identification for machine-learning-based android malware detection," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 7, pp. 3216–3225, 2018.

[5] S. Wang, Q. Yan, Z. Chen, B. Yang, C. Zhao, and M. Conti, "Textdroid: Semantics-based detection of mobile malware using network flows," in *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 18–23, IEEE, 2017.

[6] K. Xu, Y. Li, R. H. Deng, and K. Chen, "Deeprefiner: Multi-layer android malware detection system applying deep neural networks," in *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 473–487, IEEE, 2018.

[7] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones," *ACM Transactions on Computer Systems (TOCS)*, vol. 32, no. 2, pp. 1–29, 2014.

[8] C. Jindal, C. Salls, H. Aghakhani, K. Long, C. Kruegel, and G. Vigna, "Neurlux: dynamic malware analysis without feature engineering," in *Proceedings of the 35th Annual Computer Security Applications Conference*, pp. 444–455, 2019.

[9] A. Sharma and S. K. Dash, "Mining api calls and permissions for android malware detection," in *International Conference on Cryptology and Network Security*, pp. 191–205, Springer, 2014.

[10] Q. Jerome, K. Allix, R. State, and T. Engel, "Using opcode-sequences to detect malicious android applications," in *2014 IEEE International Conference on Communications (ICC)*, pp. 914–919, IEEE, 2014.

[11] W. Jung, S. Kim, and S. Choi, "Poster: deep learning for zero-day flash malware detection," in *36th IEEE symposium on security and privacy*, vol. 10, pp. 2809695–2817880, 2015.

[12] N. McLaughlin, J. Martinez del Rincon, B. Kang, S. Yerima, P. Miller, S. Sezer, Y. Safaei, E. Trickel, Z. Zhao, A. Doupé, *et al.*, "Deep android malware detection," in *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, pp. 301–308, 2017.

[15] X. Jiang, B. Mao, J. Guan, and X. Huang, "Android malware detection using fine-grained features," *Scientific Programming*, vol. 2020, 2020.

[13] 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.

[14] "Rovelli, p. ninja droid, [online] available: https://github.com/rovellipaolo/ninjadroid/.," Last visited February 2020.

[16] N. V. Duc, P. T. Giang, and P. M. Vi, "Permission analysis for android malware detection," in *The Proceedings of the 7th VAST-AIST Workshop "Research Collaboration: Review and perspective"*, 2015.

[17] J. Jung, H. Kim, D. Shin, M. Lee, H. Lee, S.-j. Cho, and K. Suh, "Android malware detection based on useful api calls and machine learning," in *2018 IEEE First International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, pp. 175–178, IEEE, 2018.

[18] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, 2010.

[19] 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)*, pp. 468–471, IEEE, 2016.

[20] "Average app size,2017 blog by brendon boshell," Last visited April 2020.

[21] Z. Yuan, Y. Lu, and Y. Xue, "Droiddetector: android malware characterization and detection using deep learning," *Tsinghua Science and Technology*, vol. 21, no. 1, pp. 114–123, 2016.

[22] X. Su, D. Zhang, W. Li, and K. Zhao, "A deep learning approach to android malware feature learning and detection," in *2016 IEEE Trustcom/BigDataSE/ISPA*, pp. 244–251, IEEE, 2016.

[23] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel, "Adversarial perturbations against deep neural networks for malware classification," *arXiv preprint arXiv:1606.04435*, 2016.

[24] 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)*, pp. 2633–2642, IEEE, 2018.

[25] S. Jaiswal, *Feature Engineering & Analysis Towards Temporally Robust Detection of Android Malware*. PhD thesis, Indian Institute of Technology Kanpur, 2019.

[26] T. Chakraborty, F. Pierazzi, and V. Subrahmanian, "Ec2: Ensemble clustering and classification for predicting android malware families," *IEEE Transactions on Dependable and Secure Computing*, 2017.

[27] Y.-l. Zhao and Q. Qian, "Android malware identification through visual exploration of disassembly files," *International Journal of Network Security*, vol. 20, no. 6, pp. 1061–1073, 2018.

[28] Y. Sun, Y. Chen, Y. Pan, and L. Wu, "Android malware family classification based on deep learning of code images.," *IAENG International Journal of Computer Science*, vol. 46, no. 4, 2019.

[29] S. Türker and A. B. Can, "Andmfc: Android malware family classification framework," in *2019 IEEE 30th International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC Workshops)*, pp. 1–6, IEEE, 2019.

[30] M. Fan, J. Liu, X. Luo, K. Chen, Z. Tian, Q. Zheng, and T. Liu, "Android malware familial classification and representative sample selection via frequent subgraph analysis," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 8, pp. 1890–1905, 2018.