On the Influence of Image Settings in Deep Learning-based Malware Detection

Francesco Mercaldo[1,2], Fabio Martinelli[2], Antonella Santone[1] and Vinod P.[3]
[1]*University of Molise, Campobasso, Italy*
[2]*Institute of Informatics and Telematics, National Research Council of Italy, Pisa, Italy*
[3]*Cochin University of Science & Technology, Kerala, India*

Keywords: Malware, Deep Learning, Security, Android, Classification.

Abstract: Considering the inadequacy of signature-based approaches for detecting malware, especially in the mobile environment, the research community is developing methodologies for detecting malware, especially using deep learning techniques, modeling applications like images. In state-of-the-art, several methods are proposed, each of one using a different kind of images and a different dimension of images: currently these are not standard settings for image preprocessing in Android malware detection. The aim of this paper is to compare different deep learning models performances to understand the best settings in terms of kind of image and image dimension. The idea is to trace a path in order to indicate the optimal settings for processing a dataset for malware detection using deep learning.

1 INTRODUCTION

Mobile devices quickly attracted the interest of the attackers, and it is easy to understand the reason why if compared with PC platforms, in our smartphones are stored more and more sensitive and private information (Gandotra et al., 2014; Fasano et al., 2019; Ciobanu et al., 2019; Martinelli et al., 2019). Furthermore, smartphones manage the SIM card in which there is our credit, also for this reason mobile surface is appealing from malicious software writer's point of view (Xiao et al., 2019).

Mobile operating systems producers tried to remedy to this rampant spread of malicious software aimed to spy infected users. For instance, Google in order to consent the publication of a new app on Play Store (the official market for Android users) requires a deep scan of the app aimed to find possible malicious activities. Indeed, the new app must be submitted to Bouncer (Oberheide and Mille, 2012), an automatic application scanning system introduced in 2012 with following distinctive features.

Bouncer performs a static analysis using the anti-malware provided by VirusTotal (a service able to evaluate the application simultaneously with 60 different anti-malware) but, considering the signature-based detection approach offered by current anti-malware technologies, it is possible to mark a malicious sample as malware only if their signature is present into the anti-malware repository (and consequently it is not possible to detect zero-day threats. (Mercaldo et al., 2016b), (Mercaldo et al., 2016a)).

With regard to the dynamic analysis, the app is ran for a limited time window (5 minutes): in case the app does not exhibit the malicious behaviour in this period passes this test (Mercaldo and Santone, 2021). Furthermore, usually malware is able to understand whether it is executed on a virtual environment (in this case it will not perform the malicious action, to avoid the detection on sandbox (Petsas et al., 2014)).

The current defense mechanism from malicious software is represented by anti-malware software, currently based on the extraction of the so-called signature. Basically the anti-malware vendor release a repository of signature, each signature is related to a specific malicious behaviour. The signature is obtained from information gathered from the samples, and they are in form of string. This mechanism has two principal drawbacks: first of all the signature extraction is a laborious and time-consuming process performed by security analysts, symptomatic that a threat to be recognized must be widespread; the second point is related to the generated signature, that is easy to circumvent also when trivial obfuscation techniques are applied, as demonstrated in (Rastogi et al., 2013; Rastogi et al., 2014) (for this reason

Mercaldo, F., Martinelli, F., Santone, A. and P., V.
On the Influence of Image Settings in Deep Learning-based Malware Detection.
DOI: 10.5220/0010890600003120
In *Proceedings of the 8th International Conference on Information Systems Security and Privacy (ICISSP 2022)*, pages 669-676
ISBN: 978-989-758-553-1; ISSN: 2184-4356

malware writers produce several variants exhibiting the same malicious behavior by applying obfuscation techniques, usually with automatic framework).

Considering the weaknesses of the defense mechanisms (for example, Bouncer and the current signature-based anti-malware technologies), researchers boosted the development of methods and tools aimed to stem this mobile malware phenomenon, but currently the main issue in the adoption of the proposed solutions is the presence of false positives.

Considering this situation, there are a plethora of techniques proposed by current state-of-the-art literature aimed to detect malware, with particular regard to the Android operating systems.

The techniques currently employed are basically based on static (i.e., the application is analysed without the need to analysed it) and dynamic (i.e., the application must be run to analyse the behaviour). With the advent of the deep learning techniques, that are mainly working of images, also in malware analysis research field, we are witnessing an increase in the adoption of applications modeled as images in order to generate deep learning models.

Most of these methods extract an image directly from the binary code of the application, some methods propose a grayscale image, others methods propose color images. Another parameter is given by the size of the input image. In this paper we propose a comparison between different deep learning models, in particular we evaluate the effectiveness of models trained with both greyscale and color images, also using different input image sizes, in order to find the ideal settings in terms of image size and type of images for the malware detection task in the Android environment.

Image based malware detection considers the representation of mobile binaries as gray-scale or RGB images, the rationale is that images related to samples belonging to the same malicious family should appear very similar in layout and texture. Contrarily, different images are expected from benign and malicious samples. Typically these techniques are applied starting from the malware binary, thus it is platform-independent i.e., there is no need to reverse engineer the application code to extract some kind of feature from the source code.

In this paper we propose a set of experiments aimed to show the best preprocessing settings. In a nutshell we design a deep learning network and we input this network with different image configurations. The main aim is to indicate the best image settings for experiments in malware detection by images in Android environment.

The paper proceeds as follows: in the next section we present the proposed deep learning network, Section 3 shows the results of the different experiments we performed, in Section 4 current state-of-the-art in mobile malware detection is discussed and, finally, in last section conclusion and future research directions are drawn.

2 METHOD

In this section we describe the proposed method aimed to detect Android malware from images, with the aim to understand the best preprocessing settings.

Figure 1 shows the workflow of the proposed approach.

To convert a binary to an image we treat the sequence of bytes representing the binary as the bytes of a gray-scale and RGB PNG image (Bhodia et al., 2019; Nataraj et al., 2011). We consider a predefined width of 256, and a variable length, depending on the size of the binary. We developed a script to encode any binary file into two different lossless (greyscale and RGB) PNG.

In detail the script, developed by authors, is based on following steps:

1. each bytes of the binary file are converted to numbers (0-255), which will then define a pixel color;
2. for each bytes will be a grayscale and RGB pixel in the final PNG image.

An implementation of the script to encode a binary file into a PNG is available at the following url: https://github.com/leeroybrun/Bin2PNG.

The most common pixel format is the byte image, where this number is stored as an 8-bit integer giving a range of possible values from 0 to 255. Typically zero is taken to be black, and 255 is taken to be white. Values inbetween make up the different shades of gray.

In Figure 2 we show the architecture of the proposed deep learning network.

The proposed network is composed by 12 different layers, in detail we exploit a combination of following layers:

- *Conv2D:* this layer represents a 2D convolution layer (e.g. spatial convolution over images). This layer creates a convolution kernel that is convolved with the layer input with the aim to produce a tensor of outputs.
- *MaxPooling2D:* Max pooling operation for 2D spatial data. Downsamples the input along its spatial dimensions (height and width) by taking the

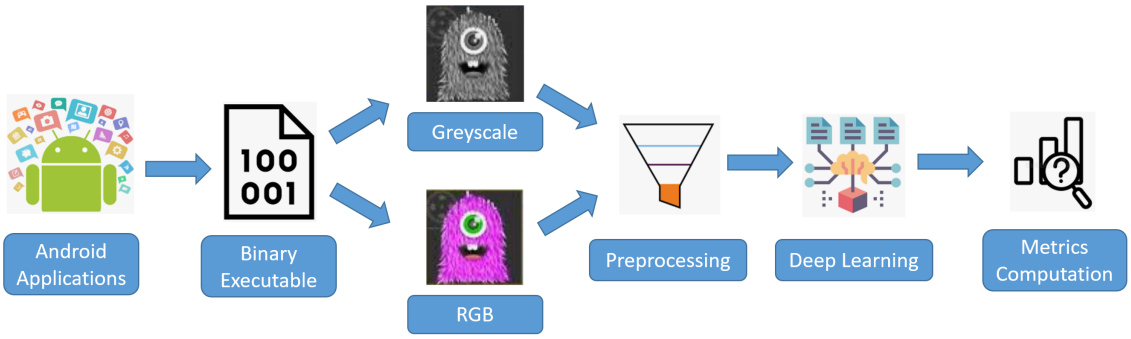

Figure 1: The proposed method.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 248, 248, 32)	320
max_pooling2d (MaxPooling2D)	(None, 124, 124, 32)	0
conv2d_1 (Conv2D)	(None, 122, 122, 64)	18496
max_pooling2d_1 (MaxPooling2	(None, 61, 61, 64)	0
conv2d_2 (Conv2D)	(None, 59, 59, 128)	73856
max_pooling2d_2 (MaxPooling2	(None, 29, 29, 128)	0
flatten (Flatten)	(None, 107648)	0
dropout (Dropout)	(None, 107648)	0
dense (Dense)	(None, 512)	55116288
dropout_1 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 256)	131328
dropout_2 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 2)	514

Total params: 55,340,802
Trainable params: 55,340,802
Non-trainable params: 0

Figure 2: The deep learning network.

maximum value over an input window (of size defined by pool_size) for each channel of the input.

- *Flatten:* The Flatten layer is used to ”flatten” the input, that is, to multidimensionalize the multidimensional input, often in the transition from the convolutional layer to the fully connected layer. Flatten does not affect the size of the batch.
- *Dropout:* Applies Dropout to the input. The Dropout layer randomly sets input units to 0 with a frequency of rate at each step during training time, which helps prevent overfitting. Inputs not set to 0 are scaled up by 1/(1 - rate) such that the sum over all inputs is unchanged.
- *Dense:* The dense layer is a neural network layer that is connected deeply, which means each neuron in the dense layer receives input from all neurons of its previous layer. The dense layer is found to be the most commonly used layer in the models. In the background, the dense layer performs a matrix-vector multiplication. The values used in the matrix are actually parameters that can be trained and updated with the help of backpropagation.

The first step, the *Training*, starting from the data (in this case the images obtained from the Android executable), an inferred function is generated. The inferred function provided by the deep learning network should be able to discriminate between different features belonging to several classes i.e., the function should define the boundaries between the numeric vectors belonging to several classes. In our case the classes are malware and trusted.

One limit about the quality of the inferred function, and therefore of the supervised machine learning techniques, is related to the training data: it is important that the data used to build models are free of bias, in order to generate models able to predict the right class for unseen instances.

With bias we are referring to applications considered in the training set but with the wrong label: this is a situation that can be conduct to classifier with poor performances (Parnas, 2017). To avoid these situations we carefully labelled the real-world Android applications considered in the analysis, as described in the Experimental Evaluation section.

The second step, once generated the models is the evaluation of their effectiveness: the *Testing* one, aimed to evaluate the performances of the model considering a set of well-known state-of-the-art metrics.

The effectiveness of the models built using the inferred functions is measured using well-know information retrieval metrics. It is important to evaluate instances not included into the training data in order to evaluate the (real) performance of the built models, in the next section more details about the evaluation are provided.

3 EXPERIMENTAL RESULTS

In this section we present the dataset we exploit to perform experiments and the experimental analysis results.

3.1 The Dataset

We built a dataset composed of 5560 trusted and 5560 malware Android applications.

The trusted applications were automatically collected from Google Play (Google, 2015), by using a script which queries an unofficial python API (pyt, 2014) to search and download applications from Android official market. The applications retrieved were among the most downloaded from different categories (call & contacts, education, entertainment, GPS & travel, internet, lifestyle, news & weather, productivity, utilities, business, communication, email & SMS, fun & games, health & fitness, live wallpapers, personalization) were downloaded from Google Play (Google, 2015) (and then controlled by Google Bouncer (Oberheide and Mille, 2012)) from January 2021 to June 2021, while malware applications of different nature and malicious intents (premium call & SMS, selling user information, advertisement, SMS spam, stealing user credentials, ransom) from Drebin Dataset (Arp et al., 2014; Spreitzenbarth et al., 2013).

We analysed the trusted dataset with the VirusTotal service (vir, 2015). This service run 52 different antivirus software (e.g., Symantec, Avast, Kasperky, McAfee, Panda, and others) on each application: the output confirmed that the trusted applications included in our dataset did not contain malicious payload.

Malware dataset is also partitioned according to the *malware family*: each family contains samples which have in common several characteristics, like payload installation, the kind of attack and events that trigger malicious payload (Zhou and Jiang, 2012). Table 1 shows the 10 malware families with the largest number of applications in our malware dataset with installation type, kind of attack and event activating malicious payload.

We briefly describe the malicious payload action for the top 10 populous families in our dataset. The dataset includes malicious payload related to 178 malicious families.

1. The samples of *FakeInstaller* family have the main payload in common but have different code implementations, and some of them also have an extra payload. FakeInstaller malware is server-side polymorphic, which means the server could provide different `.apk` files for the same URL request. There are variants of FakeInstaller that not only send SMS messages to premium rate numbers, but also include a backdoor to receive commands from a remote server. There is a large number of variants for this family, and it has distributed in hundreds of websites and alternative markets. The members of this family hide their malicious code inside repackaged version of popular applications. During the installation process the malware sends expensive SMS messages to premium services owned by the malware authors.
2. *DroidKungFu* installs a backdoor that allows attackers to access the smartphone when they want and use it as they please. They could even turn it into a bot. This malware encrypts two known root exploits, exploit and rage against the cage, to break out of the Android security container. When it runs, it decrypts these exploits and then contacts a remote server without the user knowing.
3. *Plankton* uses an available native functionality (i.e., class loading) to forward details like IMEI and browser history to a remote server. It is present in a wide number of versions as harmful adware that download unwanted advertisements and it changes the browser homepage or add unwanted bookmarks to it.
4. The *Opfake* samples make use of an algorithm that can change shape over time so to evade the antimalware. The Opfake malware demands payment for the application content through premium text messages. This family represents an example of polymorphic malware in Android environment: it is written with an algorithm that can change shape over time so to evade any detection by signature based antimalware.
5. *GinMaster* family contains a malicious service with the ability to root devices to escalate privileges, steal confidential information and send to a remote website, as well as install applica-

Table 1: Number of samples for the top 10 families with installation details (standalone, repackaging, update), kind of attack (trojan, botnet) and events that trigger malicious payload.

Family	Inst.	Attack	Activation	Apps
FakeInstaller	s	t,b		925
DroidKungFu	r	t	boot,batt,sys	667
Plankton	s,u	t,b		625
Opfake	r	t		613
GinMaster	r	t	boot	339
BaseBridge	r,u	t	boot,sms,net,batt	330
Kmin	s	t	boot	147
Geinimi	r	t	boot,sms	92
Adrd	r	t	net,call	91
DroidDream	r	b	main	81

tions without user interaction. It is also a trojan application and similarly to the DroidKungFu family the malware starts its malicious services as soon as it receives a BOOT_COMPLETED or USER_PRESENT intent. The malware can successfully avoid detection by mobile anti-virus software by using polymorphic techniques to hide malicious code, obfuscating class names for each infected object, and randomizing package names and self-signed certificates for applications.

6. *BaseBridge* malware sends information to a remote server running one ore more malicious services in background, like IMEI, IMSI and other files to premium-rate numbers. BaseBridge malware is able to obtain the permissions to use Internet and to kill the processes of antimalware application in background.
7. *Kmin* malware is similar to BaseBridge, but does not kill antimalware processes.
8. *Geinimi* is the first Android malware in the wild that displays botnet-like capabilities. Once the malware is installed, it has the potential to receive commands from a remote server that allows the owner of that server to control the phone. Geinimi makes use of a bytecode obfuscator. The malware belonging to this family is able to read, collect, delete SMS, send contact informations to a remote server, make phone call silently and also launch a web browser to a specific URL to start files download.
9. *Adrd* family is very close to Geinimi but with less server side commands, it also compromises personal data such as IMEI and IMSI of infected device. In addiction to Geinimi, this one is able to modify device settings.
10. *DroidDream* is another example of botnet, it gained root access to device to access unique identification information. This malware could also downloads additional malicious programs without the user's knowledge as well as open the phone up to control by hackers. The name derives from the fact that it was set up to run between the hours of 11pm and 8am when users were most likely to be sleeping and their phones less likely to be in use.

3.2 Performances

Below we present the experimental results we obtained for different dataset preprocessing.

Table 2 show the analysis results. For the evaluation following metrics are considered: Loss, Precision, Recall, Accuracy and Area under the curve (AUC).

We split the dataset into three parts: Training, Validation and Testing. In the training dataset there are the sample of data used to fit the model. In the validation dataset there are the sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters. The evaluation becomes more biased as skill on the validation dataset is incorporated into the model configuration. In the test dataset there are the sample of data used to provide an unbiased evaluation of a final model fit on the training dataset.

Table 2 shows the experimental results for different kind of images (Greyscale and RGB) and for different size of RGB images.

As shown from Table 2 the best performances are obtained by exploited the Greyscale images. In fact by exploiting this data format the precision, the recall and the accuracy obtained is equal to 0.99. We evaluated the classification performances exploiting RGB images of different dimensions i.e., 32x32, 64x64, 128x128 and 256x256. All the RGB experiments show a decrease in the classification performances, if compared to the ones obtained by the Greyscale images. In fact, the accuracy is ranging from 0.9818 (with RGB images with a size of 32x32) to 0.985 (using the RGB images with a dimension equal to 256x256).

4 RELATED WORK

Current state-of-the-art literature is discussed in this section.

AndroDialysis (Feizollah et al., 2017) proposes a classifier that exploits the rich semantics of *Intents*, i.e., a messaging object of the Android communication system describing the operations that an application intends to perform and that can be extracted from the code. The cited paper shows that the use of Android Intents allows obtaining a better detection ratio that the use of permissions (Canfora et al., 2013).

Methods based on static analysis do not require infecting any device nor do they require any instrumented platform for collecting data during execution in a controlled environment. On the other hand, static methods may be circumvented by code obfuscation techniques rather easily (Canfora et al., 2015; Cimitile et al., 2017; Mercaldo et al., 2016b; Martinelli et al., 2018; Mercaldo et al., 2016c).

A classification method based on features collected both statically and dynamically is proposed in (Lindorfer et al., 2015) designed a tool able to assess the maliciousness of Android applications. The authors employ a machine learning technique with the

Table 2: Experimental Results.

Model	Loss	Precision	Recall	Accuracy	AUC	val_Loss	val_Precision	val_Recall	val_Accuracy	val_AUC
Greyscale	0.0128	0.9955	0.9955	0.9955	0.9997	0.4027	0.9361	0.9361	0.9361	0.9642
Colour32	0.0428	0.9818	0.9818	0.9818	0.9985	0.5847	0.9137	0.9137	0.9137	0.9461
Colour64	0.0381	0.9854	0.9854	0.9854	0.9991	0.3953	0.9326	0.9326	0.9326	0.9654
Colour128	0.0385	0.985	0.985	0.985	0.999	0.4669	0.9396	0.9396	0.9396	0.9606
Colour256	0.0385	0.985	0.985	0.985	0.999	0.4669	0.9396	0.9396	0.9396	0.9606

aim of classifying Android applications using a feature set gathered from static and dynamic analysis of a set of known malicious and legitimate mobile applications. The BRIDEMAID (Martinelli et al., 2017; Faiella et al., 2017) framework advocates a similar hybrid approach and includes a multi-level monitoring of device, app and user behavior in order to detect malicious behaviors at run time. The Andromaly framework (Shabtai et al., 2012) proposes a classification based solely on dynamic features. The framework consists of a form of Host-based Malware Detection System able to monitor features (i.e., CPU consumption, number of packets sent through the network, number of running processes and battery level) and events obtained from the mobile device during execution.

A combination of static and dynamic techniques is used in (Blasing et al., 2010) for analyzing suspicious Android applications. The approach consists, essentially, in executing the application within a sandbox and catching all potentially dangerous events, i.e., file open operations, connections to remote server and so on. The method is assessed on an example application that implements a form of Denial of Service, i.e., a fork bomb which uses *Runtime.Exec()* to start an external binary program and creates sub-processes of itself in an infinite loop.

Several techniques for dynamic analysis have been proposed that consider *power consumption* as the discriminating feature between benign and malicious mobile applications. Authors in (Dixon et al., 2011) hypothesized that there is a strong correlation between mobile devices power consumption pattern and location and studied power consumption data from twenty smartphone users collected over a period of three months. They tested the method on a Nokia 5500 Sport to evaluate real-world mobile malware. Method proposed in reference (Kim et al., 2008) consists in a power-aware malware-detection framework monitoring and detecting unknown energy-depletion threats. Their solution is composed of (1) a power monitor which collects power samples and builds a power consumption history from the collected samples, and (2) a data analyzer which generates a power signature from the constructed history. To generate a power signature, noise-filtering and data-compression are applied, with the aim to reduce the detection overhead. They conducted the experiment using an HP iPAQ running a Windows Mobile OS.

Researchers in (Shabtai et al., 2010) detect suspicious temporal patterns to decide whether an intrusion is found, using patterns predefined by security experts. They use several features to perform this task, for instance memory, CPU and power consumption, keyboard usage and so on. Their data-set was formed by five Android malicious applications, which were developed by the authors.

The approach presented in (Ferrante et al., 2016) exploits supervised and unsupervised classification in order to identify the moment in which an application exhibits a malware behavior. Despite the general idea and aim are similar to those of this work, the cited paper lacks the visualization component and hence can hardly be used directly by the analyst.

Furthermore, several methods of static analysis have been proposed for identifying the pattern of interaction among the modules of an application and obtaining indications about possible privacy leaks. Amandroid (Wei et al., 2014) executes an ICC (i.e., the Inter-Component Communication) analysis by constructing a Data Flow Graph and a Data Dependence Graph. FlowDroid (Arzt et al., 2014) includes in the ICC analysis both the code and configuration files of the examined app, which may help in reducing both missed leaks and false positives.

5 CONCLUSION AND FUTURE WORKS

Recently, researchers from both the academic and industrial word are proposing methods to detect malware, with particular regard to the Android environment, exploiting images and deep learning. Different authors typically perform a preprocessing of the dataset using different size of images and different format (i.e., Greyscale and RGB). For this reason in this paper we input a deep learning architecture with images obtained from the same dataset of Android applications. In particular we consider two different types of images (i.e., Greyscale and RGB) and different sizes for the RGB images (i.e., 32x32, 64x64, 128x128 and 256x256). We compute different metrics to evaluate the best setting for the dataset preprocessing. From the experimental results it emerges that

the Greyscale image is able to outperform the RGB ones, regardless of the size used by the latter. As future work, we are currently investigating whether the same considering emerging from the experimental analysis we presented can be demonstrated on malware working on different operating systems i.e., iOS and Microsoft Windows.

REFERENCES

(last visit 09 April 2015). Virustotal. https://www.virustotal.com/.

(last visit 25 November 2014). Google play unofficial python api. https://github.com/egirault/googleplay-api.

Arp, D., Spreitzenbarth, M., Huebner, M., Gascon, H., and Rieck, K. (2014). Drebin: Efficient and explainable detection of android malware in your pocket. In *Proceedings of 21th Annual Network and Distributed System Security Symposium (NDSS)*.

Arzt, S., Rasthofer, S., Fritz, C., Bodden, E., Bartel, A., Klein, J., Le Traon, Y., Octeau, D., and McDaniel, P. (2014). Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. *ACM SIGPLAN Notices*, 49(6):259–269.

Bhodia, N., Prajapati, P., Di Troia, F., and Stamp, M. (2019). Transfer learning for image-based malware classification. *arXiv preprint arXiv:1903.11551*.

Blasing, T., Schmidt, A.-D., Batyuk, L., Camtepe, S. A., and Albayrak, S. (2010). An android application sandbox system for suspicious software detection. In *Proceedings of 5th International Conference on Malicious and Unwanted Software*.

Canfora, G., Di Sorbo, A., Mercaldo, F., and Visaggio, C. A. (2015). Obfuscation techniques against signature-based detection: a case study. In *2015 Mobile Systems Technologies Workshop (MST)*, pages 21–26. IEEE.

Canfora, G., Mercaldo, F., and Visaggio, C. A. (2013). A classifier of malicious android applications. In *Proceedings of the 2nd International Workshop on Security of Mobile Applications, in conjunction with the International Conference on Availability, Reliability and Security*.

Cimitile, A., Mercaldo, F., Nardone, V., Santone, A., and Visaggio, C. A. (2017). Talos: no more ransomware victims with formal methods. *International Journal of Information Security*, pages 1–20.

Ciobanu, M. G., Fasano, F., Martinelli, F., Mercaldo, F., and Santone, A. (2019). A data life cycle modeling proposal by means of formal methods. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, pages 670–672. ACM.

Dixon, B., Jiang, Y., Jaiantilal, A., and Mishra, S. (2011). Location based power analysis to detect malicious code in smartphones. In *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*.

Faiella, M., La Marra, A., Martinelli, F., Mercaldo, F., Saracino, A., and Sheikhalishahi, M. (2017). A distributed framework for collaborative and dynamic analysis of android malware. In *2017 25th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pages 321–328. IEEE.

Fasano, F., Martinelli, F., Mercaldo, F., and Santone, A. (2019). Energy consumption metrics for mobile device dynamic malware detection. *Procedia Computer Science*, 159:1045–1052.

Feizollah, A., Anuar, N. B., Salleh, R., Suarez-Tangil, G., and Furnell, S. (2017). Androdialysis: analysis of android intent effectiveness in malware detection. *computers & security*, 65:121–134.

Ferrante, A., Medvet, E., Mercaldo, F., Milosevic, J., and Visaggio, C. A. (2016). Spotting the malicious moment: Characterizing malware behavior using dynamic features. In *Availability, Reliability and Security (ARES), 2016 11th International Conference on*, pages 372–381. IEEE.

Gandotra, E., Bansal, D., and Sofat, S. (2014). Malware analysis and classification: A survey. *Journal of Information Security*, 5(02):56.

Google (2015). Play. https://play.google.com/store.

Kim, H., Smith, J., and Shin, K. G. (2008). Detecting energy-greedy anomalies and mobile malware variants. In *Proceedings of the 6th international conference on Mobile systems, applications, and services*.

Lindorfer, M., Neugschwandtner, M., and Platzer, C. (2015). Marvin: Efficient and comprehensive mobile app classification through static and dynamic analysis. In *Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual*, volume 2, pages 422–433. IEEE.

Martinelli, F., Mercaldo, F., Nardone, V., Santone, A., Sangaiah, A. K., and Cimitile, A. (2018). Evaluating model checking for cyber threats code obfuscation identification. *Journal of Parallel and Distributed Computing*, 119:203–218.

Martinelli, F., Mercaldo, F., and Santone, A. (2019). Social network polluting contents detection through deep learning techniques. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–10. IEEE.

Martinelli, F., Mercaldo, F., and Saracino, A. (2017). Bridemaid: An hybrid tool for accurate detection of android malware. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 899–901. ACM.

Mercaldo, F., Nardone, V., and Santone, A. (2016a). Ransomware inside out. In *Availability, Reliability and Security (ARES), 2016 11th International Conference on*, pages 628–637. IEEE.

Mercaldo, F., Nardone, V., Santone, A., and Visaggio, C. A. (2016b). Hey malware, i can find you! In *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2016 IEEE 25th International Conference on*, pages 261–262. IEEE.

Mercaldo, F., Nardone, V., Santone, A., and Visaggio, C. A. (2016c). Ransomware steals your phone. formal methods rescue it. In *International Conference on Formal Techniques for Distributed Objects, Components, and Systems*, pages 212–221. Springer.

Mercaldo, F. and Santone, A. (2021). Audio signal processing for android malware detection and family identification. *Journal of Computer Virology and Hacking Techniques*, 17(2):139–152.

Nataraj, L., Karthikeyan, S., Jacob, G., and Manjunath, B. (2011). Malware images: visualization and automatic classification. In *Proceedings of the 8th international symposium on visualization for cyber security*, page 4. ACM.

Oberheide, J. and Mille, C. (2012). Dissecting the android bouncer. In *SummerCon*.

Parnas, D. L. (2017). The real risks of artificial intelligence. *Communications of the ACM*, 60(10):27–31.

Petsas, T., Voyatzis, G., Athanasopoulos, E., Polychronakis, M., and Ioannidis, S. (2014). Rage against the virtual machine: hindering dynamic analysis of android malware. In *Proceedings of the Seventh European Workshop on System Security*, page 5. ACM.

Rastogi, V., Chen, Y., and Jiang, X. (2013). Droidchameleon: evaluating android anti-malware against transformation attacks. In *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*, pages 329–334. ACM.

Rastogi, V., Chen, Y., and Jiang, X. (2014). Catch me if you can: Evaluating android anti-malware against transformation attacks. *IEEE Transactions on Information Forensics and Security*, 9(1):99–108.

Shabtai, A., Kanonov, U., and Elovici, Y. (2010). Intrusion detection for mobile devices using the knowledge-based, temporal abstraction method. In *Journal of Systems and Software, 83(8):1524–1537*.

Shabtai, A., Kanonov, U., Elovici, Y., Glezer, C., and Weiss, Y. (2012). "andromaly": a behavioral malware detection framework for android devices. *Journal of Intelligent Information Systems*, 38(1):161–190.

Spreitzenbarth, M., Echtler, F., Schreck, T., Freling, F. C., and Hoffmann, J. (2013). Mobilesandbox: Looking deeper into android applications. In *28th International ACM Symposium on Applied Computing (SAC)*.

Wei, F., Roy, S., Ou, X., et al. (2014). Amandroid: A precise and general inter-component data flow analysis framework for security vetting of android apps. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 1329–1341. ACM.

Xiao, X., Zhang, S., Mercaldo, F., Hu, G., and Sangaiah, A. K. (2019). Android malware detection based on system call sequences and lstm. *Multimedia Tools and Applications*, 78(4):3979–3999.

Zhou, Y. and Jiang, X. (2012). Dissecting android malware: Characterization and evolution. In *Proceedings of 33rd IEEE Symposium on Security and Privacy (Oakland 2012)*.