Cluster Computing (2022) 25:2487–2500
https://doi.org/10.1007/s10586-021-03490-2

A federated approach to Android malware classification through Perm-Maps

Gianni D'Angelo[1] · Francesco Palmieri[1] · Antonio Robustelli[1]

Received: 29 June 2021 / Revised: 13 October 2021 / Accepted: 24 November 2021 / Published online: 4 February 2022
© The Author(s) 2022

Abstract
In the last decades, mobile-based apps have been increasingly used in several application fields for many purposes involving a high number of human activities. Unfortunately, in addition to this, the number of cyber-attacks related to mobile platforms is increasing day-by-day. However, although advances in Artificial Intelligence science have allowed addressing many aspects of the problem, malware classification tasks are still challenging. For this reason, the following paper aims to propose new special features, called permission maps (Perm-Maps), which combine information related to the Android permissions and their corresponding severity levels. Such features have proven to be very effective in classifying different malware families through the usage of a convolutional neural network. Also, the advantages introduced by the Perm-Maps have been enhanced by a training process based on a federated logic. Experimental results show that the proposed approach achieves up to a 3% improvement in average accuracy with respect to J48 trees and Naive Bayes classifier, and up to 16% compared to multi-layer perceptron classifier. Furthermore, the combined use of Perm-Maps and federated logic allows dealing with unbalanced training datasets with low computational efforts.

Keywords Federated approach · Android classification · Perm-Maps · Deep neural network · Android permissions

1 Introduction

Since Android-based devices are used by thousands of end-users every year, more and more malicious applications are continuously developed by cyber-criminals in order to steal sensitive information and conduct hostile activities. According to McAfee Mobile Threat Report, in 2019, cyber-criminals have increased the effectiveness of their mobile attacks with the support of a wide variety of methods and new approaches, such as backdoors and cryptocurrencies, by making them hard to be identified and removed [32]. In addition to this, as show in Fig. 1, G DATA and McAfee experts have counted more than 4.18 million new malicious applications in 2019 [17], while Kaspersky and TechCrunch have estimated that there will be over 6 billion smartphone users worldwide by 2020 [22, 41].

Therefore, to face the following security trend and support researchers in addressing the malware detection tasks, several approaches based on machine learning (ML) and deep learning (DL) have proved to be effective in facing many aspects related to Android threats, especially when they have been combined with static and dynamic features directly extracted from mobile apps [16, 21, 31]. However, due to the continuous release of new Android malware, the related classification tasks are still challenging. As a consequence, many state-of-the-art approaches suffer from problems related to their dynamic re-training, as well as the updating training datasets.

To address these issues, in this paper, we propose new special features, called permission maps (Perm-Maps), which combine information related to the Android permissions and their corresponding severity levels. Such features are employed to classify different malware families through the usage of a convolutional neural network

✉ Antonio Robustelli
arobustelli@unisa.it

Gianni D'Angelo
giadangelo@unisa.it

Francesco Palmieri
fpalmieri@unisa.it

[1] Dipartimento Di Informatica, Università Degli Studi Di Salerno, Salerno, Italy

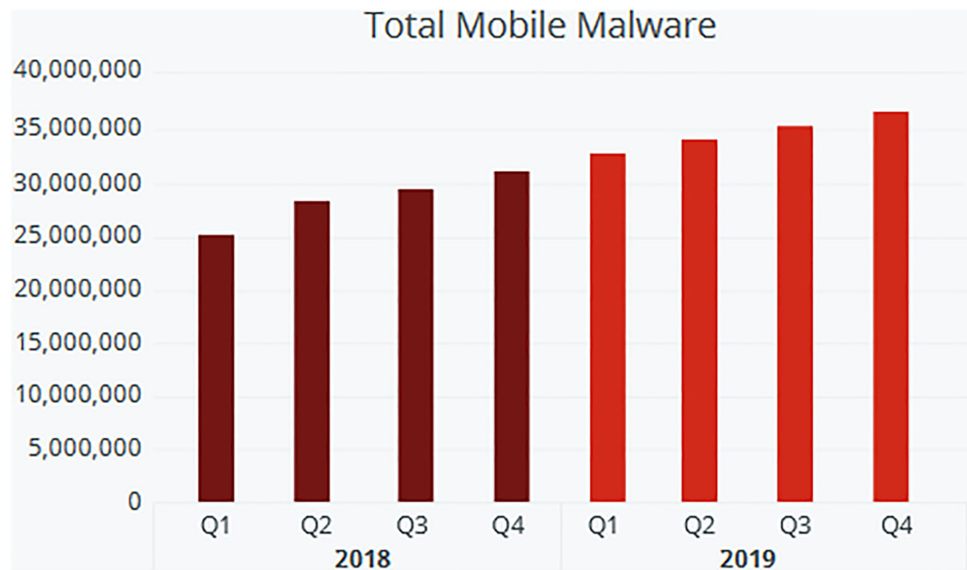


Fig. 1 Total mobile malware detections by quarter in 2018 and 2019 [32]

(CNN). Also, the advantages introduced by the Perm-Maps are being enhanced by a training process based on the federated logic, where end-user devices extract static features locally and send them to a centralized server devoted to training the employed neural network.

Next, we explore the effectiveness of the proposed Perm-Maps by comparing them with the most popular state-of-the-art ML- and DL-based approaches. Finally, to reduce the computational effort respectively required by the Perm-Maps generation and CNN training processes, we investigate a feature selection technique based on the most frequent Android permissions.

The main contributions of this paper can be summarized as follows:

1. Novel features, called Perm-Maps, are proposed to combine the Android permissions and their corresponding security levels into an image.
2. A federated architecture is presented to support the training phase of the Perm-Maps.
3. A CNN is employed to classify several Android malware families and then compared with the most popular state-of-the-art approaches.
4. A feature selection technique based on the most frequent Android permissions is investigated to reduce the computational effort required by the Perm-Maps generation and CNN training processes, respectively.

The rest of the paper is organized as follows. Section 2 will present the related works about malware classification methods for Android devices. Section 3 will report a background overview on Android permissions. Section 4 will show the definition of Perm-Map, which is based on the Android permissions and their corresponding severity levels. Section 5 will present the employed federated architecture. Section 6 will discuss the obtained results related to the proposed CNN and the investigated feature selection technique, respectively. Finally, Sect. 7 will show the conclusions and future works.

2 Related works

Since Android malware applications are continuously released every year by cyber-criminals, many detection frameworks based on static and dynamic methodologies have been proposed [16, 21, 31]. Static techniques can acquire the behaviour of the analyzed applications by performing several reverse engineering steps, and consequently, by extracting useful signatures without executing the application. For instance, Onwuzurike et al. [34] presented MaMaDROID, a new Android malware detection solution that can check the sequences of API calls associated with the activity of a mobile application.

However, static approaches are often adversely affected by the use of obfuscation techniques, and additionally, they become ineffective against polymorphic malware which is able to modify itself. This is the reason why any signature-based detection techniques are ineffective, and consequently, they are often substituted by dynamic approaches, which are based on dynamic analysis techniques, and hence, are able to analyze the behaviour of an application at run time. In 2018, Sruthi et al. [40] proposed a malware detection technique, in Windows OS environment, based on API calls. Furthermore, several works have adopted ML and DL techniques based on both static and dynamic features [14, 33, 48].

In 2016, Kolosnjaji et al. [24] investigated a comparison among different deep neural networks (DNNs) typologies.

In particular, they proposed a convolutional long short term memory (Conv-LSTM) network able to achieve an 89.0% in average accuracy, by considering 10 different Android malware categories. Kumar et al. [25] proposed a comparison among the three famous ML-based methods to detect Android malware by analyzing the visual representation of APK files formatted as Grayscale, RGB, CMYK, and HSL images, without any code extraction and decompiling operations. More precisely, they investigated the proposed technique by using decision trees (DT), Random Forest (RF), and k-nearest neighbor (k-NN), respectively. The obtained results have shown that RF is able to achieve a 91% accuracy by considering APK files formatted as Grayscale images.

In 2017 Vinayakumar et al. [42] investigate different LSTM neural networks to classify the APK files as either benign or malicious. In particular, they proposed an LSTM network able to achieve an 89.7% accuracy, by taking into account Android permissions translated as numerical information.

In 2018 Li et al. [27] proposed a comparison among different DNNs configurations based on static information, like permissions and Java code. More precisely, they compared ten distinct neural network configurations by achieving an average accuracy between 95 and 97% in the Android malware classification task. Xie et al. [47] proposed a tool called RepassDroid, which is able to classify Android applications, as benign or malicious, based on permission and Java methods. Additionally, they explored a comparison among different ML-based approaches like DT, RF, k-NN, Naive Bayes (NB), and support vector machines. The achieved results have proven that RF is able to achieve a 99.7% accuracy by taking into account 24,288 Android applications.

In 2019, Li et al. [26] proposed a novel and highly reliable DNN classifier for Android malware detection based on the extraction of several features from manifest files and source code. In particular, they considered seven different static features like app components, hardware features, permissions, intent filters, restricted and suspicious Java methods, and used permissions. Thus, they have been used to train a DNN able to obtain a 99.25% average accuracy. D'Angelo et al. [13] proposed a deep sparse autoencoders (AEs) to classify Android-based malware and goodware (GW) applications downloaded from several app stores. More precisely, they proposed a new API methods representation technique named API-images, and then, an average accuracy of 95% has been achieved by employing deep sparse AEs.

In 2020, Aonzo et al. [7] presented BAdDroIds, a mobile application that leverages DL for detecting malware on resource-constrained devices. In particular, the proposed application has been compared with the most notable Android malware detection frameworks by achieving a 98% average accuracy.

Finally, in 2021, D'Angelo et al. [12] proposed a CNN and a recurrent neural network (RNN), based on API-images, in order to classify different malware families. More precisely, they used both neural networks on five malware families on the Unisa malware dataset (UMD) by achieving 99% in average accuracy.

3 Background

In this section, some key concepts related to Android permissions and federated environments are discussed in order to understand and appreciate the novelties of the proposed approach.

3.1 Permission's overview

Android permissions can be categorized into three main typologies: **Install-time**, **Runtime**, and **Special** [4]. Install-time permissions grant an application limited access to restricted data, and thus, they allow an application to perform restricted actions that minimally affect the system or other apps. When a developer declares install-time permissions, the system automatically grants the required permissions without notifying the end-user. There are two types of Install-time permissions respectively called **normal permissions** and **signature permissions**:

- *Normal permissions* allow access to data and actions that present minimal risk for the system or end-users privacy. They can be used or identified through a protection level's value set to *normal*.
- *Signature permissions* since they are defined in another Android application, the signature permissions are granted only if the requesting and declarant applications are signed through the same certificate. Also, they can be used or identified through a protection level value set to *signed*.

Runtime permissions, also known as **dangerous permissions**, grant an application additional access to restricted data by allowing it to perform actions that substantially affect the system and other apps. When an Android application requests runtime permissions, the system presents a prompt and waits that is granted or not by the end-user. Runtime permissions can be used or identified through a protection level value set to *dangerous*.

Finally, the special permissions can be only defined by the original equipment manufacturers (OEMs) to provide access control concerning several energy-intensive actions, such as access to other applications. More precisely, they are closely associated with an app operation (app op)

related to access control, and they can be used or identified through a protection level value set to *appop*.

4 Permission maps

Although most of the techniques used in literature include both static and dynamic approaches, the static one is the most desired because it can analyze applications without running them. Accordingly, we propose new features, called Perm-Maps, derived by the malware static analysis. More precisely, A Perm-Map is a sparse matrix where Android permissions, and their corresponding severity levels, are related as fixed points and reported in an x–y plane. As depicted in the following, the proposed Perm-Maps are able to address three main issues: (i) Android malicious developers could define custom permissions to perform several hostile activities, like theft of sensitive data or launch of cyber-attacks [1]; (ii) since default and custom permissions are associated to different severity levels, also called *protection levels* or *flags*, like: normal, signature, dangerous, or their combinations, an application could be characterized by many permissions and severity levels [3, 5]. Therefore, a malicious developer could define some low severity level permissions to perform several actions without notifying the end-user; (iii) since Perm-Maps represents static features only extracted from the manifest file, they cannot be influenced by the most famous obfuscator tools, like DexGuard [18], ProGuard [19], and Obfuscapk [6].

4.1 Perm-Map creation workflow

The creation of a Perm-Map consists mainly in the following four steps:

1. Extraction of the Android permissions and their corresponding protection level.
2. Assignment of an identifier (ID^p) to any Android permission.
3. Assignment of an identifier (ID^s) to any severity level.
4. Creation of the Perm-Maps by using pairs of IDs (ID^p; ID^s) as coordinates of fixed points in an x–y plane.

The first step is accomplished by using several tools or libraries devoted to the malware static analysis. A typical approach could envisage a dictionaries creation process of the well-known Android permissions, and their protection levels, by finding them from the official documentation [2]. Alternatively, the ⟨**permission**⟩ tag can be employed to know the protection level of custom permissions. This approach is adopted by several most famous reverse engineering tools, like **Androguard** [15]. More precisely, for each permission declared into the AndroidManifest file, it is able to obtain the corresponding protection level by checking if the considered permission is known; assign a dangerous protection level otherwise.

Next, the second and third steps are accomplished by creating two dictionaries to respectively translate each Android permission and each corresponding severity level into a unique ID number. Finally, for each analyzed application, the fourth step is conducted by considering each pair of ID numbers (ID^p; ID^s) as coordinates of a fixed point, and consequently, storing the translated information in a sparse matrix. For instance, let $p1$ and $p2$ two Android permissions, and let $s3$ and $s2$ their security level, respectively. We can consider two pair of coordinates $C1 = (p1, s3)$ and $C2 = (p2, s2)$ and draw two points in an x–y plane, where axes x and y reports permissions and severity levels, respectively. However, since security levels could be different among them, it is possible to use different colour scales (like RGB or Gray-scale) to remark these differences. Figure 2 shows the complete workflow to obtain a Perm-Map.

5 A federated architecture

Since millions of Android-based applications are released every year, managing related data for model training purposes is a process that requires significant efforts, mainly associated to accessing, searching, and updating them. To overcome these issues, we present a federated architecture to support Android classification tasks through the proposed Perm-Maps. Federated architectures are based on a federated data production logic, which implies that the participating devices send their own pre-processed permission data to a centralized infrastructure devoted to provide collection services and classification-model construction and to share related information [23]. Due to its great success, the federated logic has been investigated, in the last decade, to face main issues related to the convergence process among edge and cloud infrastructures, such as data aggregation, data mobility, and services migration [10, 30, 38]. Also, it has been involved in many other famous application domains, such as cryptography solutions to preserve data security [36], optimization frameworks for the medical of things devices [37], and vehicular networks optimization [43].

In detail, the proposed architecture aims to provide a data aggregation workflow where federated devices are used as decentralized permission data sources and preliminary processing units. Additionally, a central server is employed to collect data, and then construct, share and update a classification model to be transferred as an update to each federated device, and thus, to propose a managing strategy for the involved permissions data. Therefore, the

Textual Information

```
['android.permission.WRITE_SETTINGS', 'dangerous', 'android.permission.SEND_SMS', 'dangerous', 'a
['android.permission.CALL_PHONE', 'dangerous', 'android.permission.EXPAND_STATUS_BAR', 'normal',
['android.permission.WRITE_SETTINGS', 'dangerous', 'android.permission.SEND_SMS', 'dangerous', 'a
['android.permission.SET_WALLPAPER', 'normal', 'android.permission.READ_PHONE_STATE', 'dangerous'
```

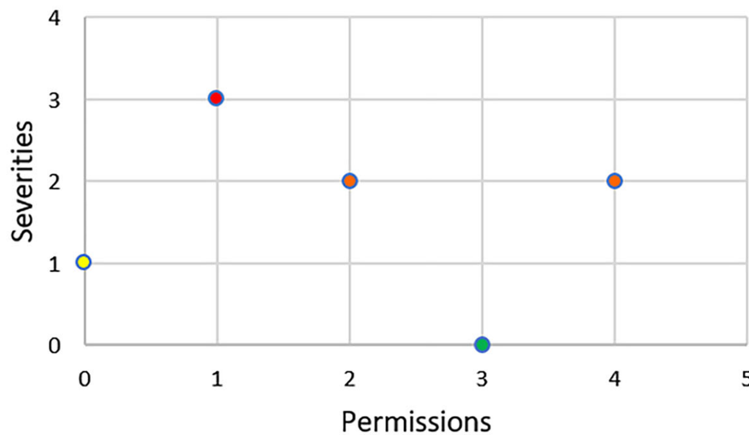
Dictionaries Creation

Permission	ID
WRITE_SETTINGS	0
CALL_PHONE	1
SET_WALLPAPER	2
...	...

Information Mapping

Severity	ID
dangerous	0
normal	1
signed	2
...	...

Perm-Maps Creation



C0 = (p0,s1) = (0,1)
C1 = (p1,s3) = (1,3)
C2 = (p2,s2) = (2,2)
C3 = (p3,s0) = (3,0)
C4 = (p4,s2) = (4,2)

Fig. 2 Perm-Maps workflow

discussed architecture works through two steps respectively named **model creation process** and **model update process**, while its main contributions can be summarized as follows:

1. A data aggregation's workflow is presented to collect data from federated devices.
2. A centralized dataset is employed to create a shared DNN model based on Perm-Maps.
3. A data update workflow is discussed to manage centralized data and re-adapt the shared model.

5.1 Model creation process

At beginning of the model creation process, each device decompresses the APK file and sends the AndroidManifest file to the central server. Thus, when data are completely stored, it will perform the Perm-Maps creation process by

following the workflow shown in Fig. 2. Basically, the server will run the CNN's training and testing phase and send the classification model to each device. Finally, each end-user will receive a notification concerning the classification result of the analyzed application. Figure 3 shows the discussed process, while its main steps can be summarized as follows:

1. End devices decompress the APK file.
2. They also send the manifest file to the central server.
3. The server runs the Perm-Maps creation process, when data are completely available.
4. It then runs the CNN's training and testing phase.
5. The server sends the classification model to each device.
6. The end devices notify the end-users about the classification result.

Note that, when an end device receives the first classification model information, it becomes able to autonomously create its Perm-Maps, and hence perform classification, without affecting the central server.

5.2 Model update process

The following phase is responsible for collecting new data when the end-user tries to install a new application. At a high level, it differs from the previous process in three main aspects:

1. If an application is unknown, it automatically stores the related manifest file on the central server.
2. If an application is unknown, it considers the end-users feedback to generate a classification label.
3. If a threshold value is reached, it trains and shares an updated model by considering new data.

Therefore, when an end-user installs an application, the device decompresses the APK, extracts the Perm-Map by reading the AndroidManifest file, and uses the classification model to make a classification. If the application is known, the classification module will notify the end-user by showing the achieved prediction. Otherwise, it will ask if the installed application is known or trusted, and subsequently, will send the manifest file and the user's answer to the central server. Thus, the employed server stores new data and, when the dataset size will have reached a threshold value, it will re-perform the Perm-Maps creation process. Finally, the server will re-run the training and testing phase and sends the updated model to each device. Figure 4 shows the discussed process, while the main steps can be summarized as follows:

1. End devices decompress the APK file.
2. They also extract the Perm-Map from the manifest file.
3. End devices also try to obtain a prediction and ask if the analyzed application is known or trusted.
4. They send the manifest file and user's answer to the server.
5. The server stores new data.
6. It then re-runs the Perm-Maps creation process, when the dataset size reaches a threshold value.
7. It also re-runs the CNN's training and testing phase.
8. Finally it sends the updated model to each device.

6 Experimental results

The first goal of experiments, reported in this section, is devoted to demonstrating the contribution of the proposed approach concerning the classification of several Android applications. Instead, the second one exploring the

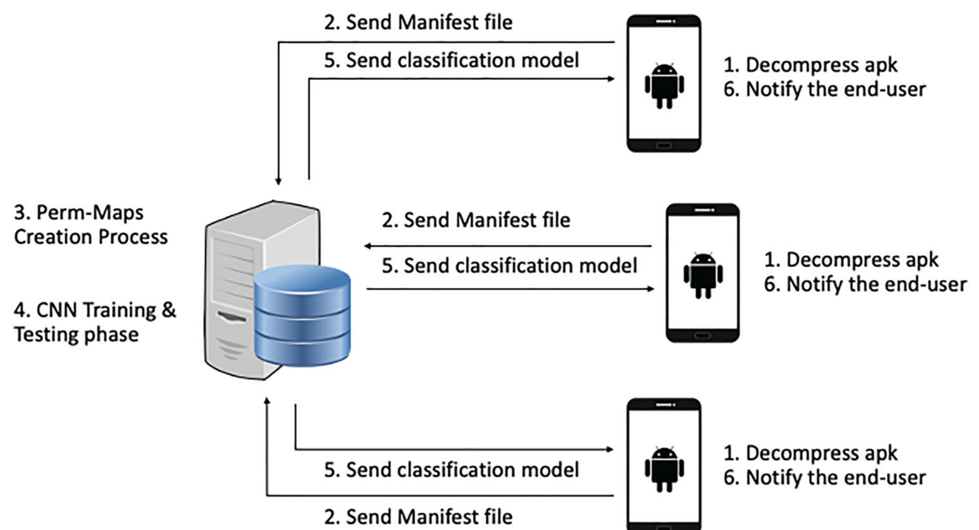


Fig. 3 Model creation process

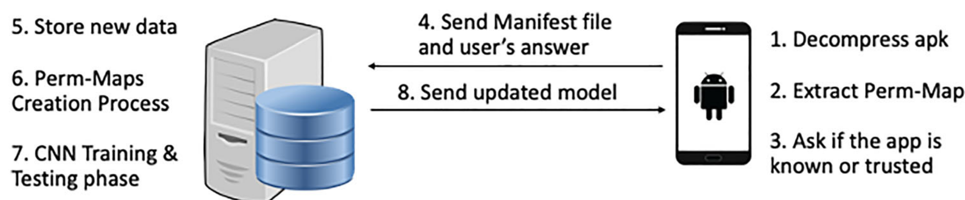


Fig. 4 Model update process

effectiveness of a feature selection technique, based on the most frequent permissions, to reduce the computational effort required by the generation and training processes of the Perm-Map and CNN, respectively.

6.1 UMD cleaning

In 2021 we developed a new Android malware dataset (AMD) called Unisa malware dataset (UMD)[1] [12] that contains 25,275 mobile applications collected by analyzing two famous datasets: AMD [28, 44] and Drebin [8, 39]. This first version of UMD consists of two main directories called **amd-cuckoo-family** and **drebin-cuckoo-family** that contain 66 and 143 Android malware families, respectively. Additionally, it provides, for each analyzed application, the report files obtained through CuckooDroid Sandbox [11, 20]. Table 1 shows an overview of the first release of UMD.

Table 1 Overview on the first version of UMD

	Analyzed APK	Families	Dimension (GB)
AMD	20,426	66	100.08
Drebin	4849	143	17.55
Total	25,275	209	117.63

In this work, we use a cleaned version of UMD (UMD-v2) obtained by applying the following modifications:

1. Consider the two main folders as a single one.
2. Merge the common families.
3. For each common family, remove the duplicates.
4. Remove each application which has got one or more malformed files.
5. Remove each application which has got one or more missing files.

The application of points (1) and (2) have reduced the number of considered families from 209 to 185. Instead, the application of points (3), (4) and (5) have reduced the number of the analyzed applications from 25,275 to 24,285. Additionally, the application of the entire protocol has reduced the dimensions (Dim.) from 117.63 to 112.45 GB. Table 2 reports a comparison between the two versions of our datasets.

6.2 Proof of concept experimental setting

We built our proof of concept testing framework within a virtualization scenario based on VirtualBox. For this work, we considered 10 categories of Android applications. In particular, the entire dataset used for training has been composed by choosing nine malware families from UMD-v2 and selecting GW applications from the following online stores: ApkPure, GooglePlay, and PlayDrone. Hence, to simulate the discussed Model Creation Process, each application has been analyzed through the **Android device cross-platform** mode of CuckooDroid [11, 20]. More precisely, in our proof of concept framework we used two Android guest virtual machines, simulating end devices, to decompress each APK file and send the AndroidManifest file to the server virtual machine. Thus, we extracted Perm-Maps by using a dedicated Python script executed on the server machine. We stored each Perm-Map as a matrix 4×298 in accordance with the maximum number of distinct severity levels and Android permissions observed, respectively. Figure 5 shows the application's distribution extracted by performing an exploratory data analysis, EDA [35, 46], and it highlights the unbalanced behaviour of the employed dataset.

Subsequently, we have split the following dataset in order to run the experiments. To this purpose, the whole dataset has been subdivided into two mutually exclusive subsets called learning and testing dataset, respectively. We used 70% of the entire dataset for learning and the remaining 30% for testing. Then, the K-fold cross-validation algorithm, with k = 10 (as recommended in [9], has been used to tune the hyper-parameters and provide an unbiased evaluation of each employed CNN. Finally, each CNN has been trained on each training set and evaluated on the corresponding testing set. Table 3 reports the main information about the involved dataset.

6.3 Proposed network and evaluation metrics

The employed CNN architecture has been developed as a sequence of two Conv2D layers with kernel_size = (2, 2), activation = relu, and no pooling. For the first one, we used 8 filters and strides = (2, 2), while for the second one we

[1] http://antlab.di.unisa.it/malware/.

Table 2 Comparison between the versions of UMD

	Analyzed APK	Families	Dimension (GB)
UMD-v1	25,275	209	117.63
UMD-v2	24,285	185	112.45

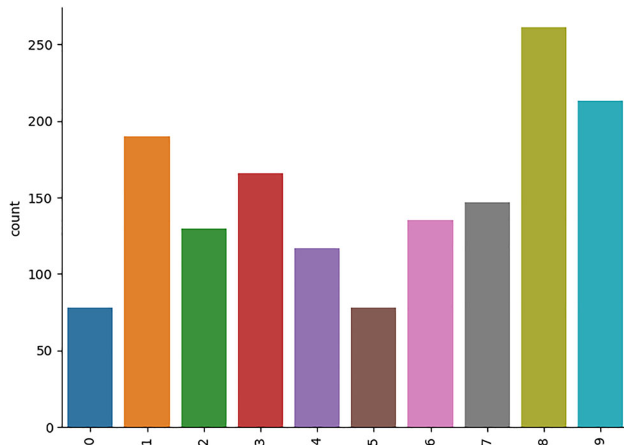


Fig. 5 Data distribution for each category

Table 3 Summary of the involved dataset

	Num. APK	Training	Testing
Adrd	78	56	22
Boqx	190	135	55
FakeDoc	130	99	31
Fusob	166	106	60
G.Master	117	82	35
GW	78	55	23
Iconosys	135	101	34
Kmin	147	104	43
Lotoor	261	183	78
Mseg	213	145	68
Total	1515	1066	449

used 2 filters and strides = (1, 1). Subsequently, we added a flatten layer to convert the latent space, from the second Conv2D layer, as a flattened sequence to fed a fully-connected softmax neural network. Therefore, 2 dense layers with 128 nodes, activation = relu, and dropout = 0.5, have been connected. Finally, a dense layer with 10 nodes and activation = softmax has been used as the output layer. Figure 6 shows the architecture of the proposed network. Additionally, the following architecture has been derived by varying the following hyper-parameters:

- numConvLayers: the number of Conv2D layers considered (1, 2, 3);
- numDenseLayers: the number of dense layers considered (1, 2, 3, 4);
- filters: the number of filters considered for each Conv2D layer (2, 4, 8, 16);
- neurons: the number of neurons considered for each dense layer (10, 32, 64, 128, 256);
- activation: activation functions employed (relu, softmax);
- strides: the stride length for each Conv2D layer (1, 2, 4);
- batch_size: considered batch_size values (16, 32, 64, 128);
- loss: loss functions used (Categorical_Crossentropy, SparseCategoricalFocalLoss).

To evaluate the classification quality of the employed neural network, the following metrics have been computed: accuracy (Acc.), sensitivity (Sens.), specificity (Spec.), precision (Prec.), area under the ROC curve (AUC), and F-measure (F-Meas or F-score). More precisely, they have been derived from a multi-class confusion matrix where, for each category, TPs (true positives) are the applications correctly classified, TNs (true negatives) are the applications correctly classified in another category, FPs (false positives) are the applications incorrectly identified as a considered category, while FNs (false negatives) are the applications in another category incorrectly identified as a considered category. Subsequently, in order to obtain a global validation, the average values (Avg.) among all metrics have been computed.

6.4 Achieved results

The proposed CNN has been trained and tested on an iMac equipped with an Intel 6-Core i7 CPU @ 3.20 GHz, and 16 GB RAM. The employed neural network has been compiled with Adam optimizer and SparseCategoricalFocalLoss function [29], which is a useful function to fit neural networks in presence of unbalanced datasets. Then, it has been trained with batch_size = 64, and 150 epochs by using the 70/30 criteria and the K-fold cross-validation algorithm with k = 10. We chose the following hyper-parameters according to the achieved results from the testing process. Tables 4 and 5 show results that have been obtained from the testing phase by respectively using the 70/30 criteria and the K-fold cross-validation algorithm with k = 10, while Table 6 shows the multi class confusion matrix related to the 70/30 criteria.

Furthermore, to face the yearly growth of the malicious applications and analyze the update process of the

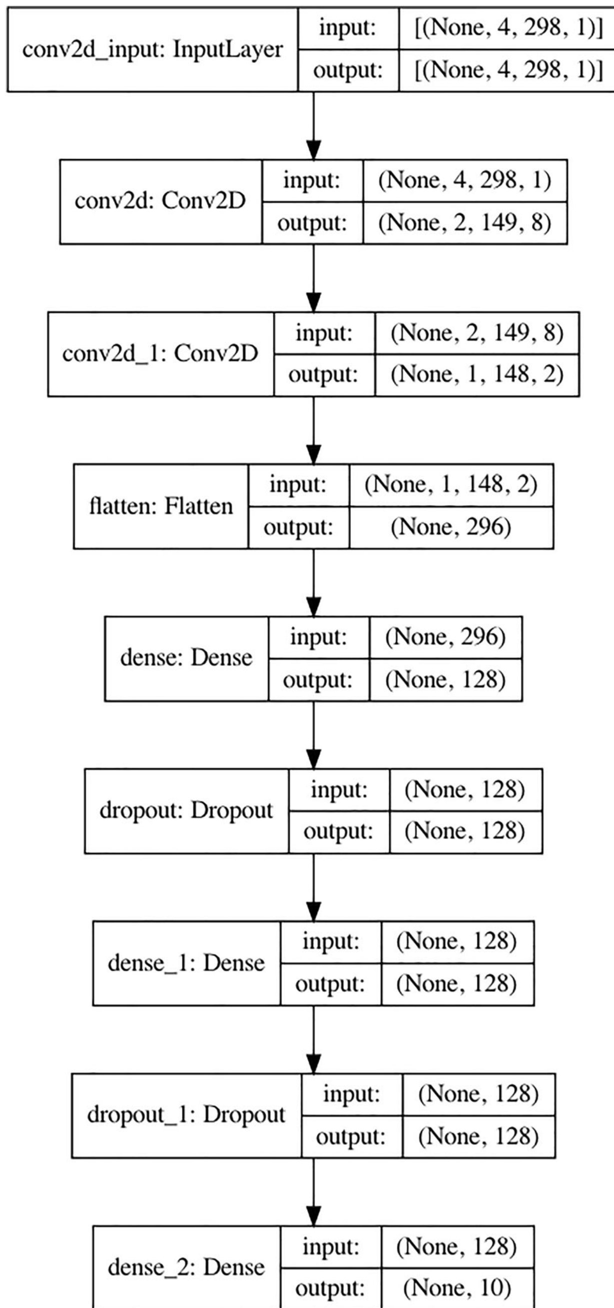


Fig. 6 Architecture of the employed neural network

presented architecture, we have estimated the data growth range within which to readjust the proposed CNN. More precisely, we have reduced the whole dataset by 5% through an iterative process. At each step, 5% of data have been randomly removed, and thus, we have employed the considered sub-dataset to train and test the proposed CNN by following the 70/30 criteria. Table 7 summarizes the classification metrics derived by the testing phase for each considered sub-dataset.

Table 4 Performance metrics related to 70/30 criteria

	Acc.	Spec.	Prec.	Sens.	F-score	AUC
Adrd	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
Boqx	0.9912	0.9898	0.9322	1.0000	0.9649	0.9949
FakeDoc	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
Fusob	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
G.Master	0.9978	0.9976	1.0000	0.9722	0.9859	0.9849
GW	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
Iconosys	0.9978	1.0000	1.0000	0.9714	0.9855	0.9857
Kmin	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
Lotoor	0.9867	0.9945	0.9750	0.9512	0.9630	0.9728
Mseg	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
Avg.	0.9974	0.9982	0.9906	0.9895	0.9898	0.9937

Table 5 Performance metrics related to K-fold k = 10

	Acc.	Spec.	Prec.	Sens.	F-score	AUC
Fold 1	0.9960	0.9977	0.9830	0.9757	0.9789	0.9867
Fold 2	0.9956	0.9974	0.9839	0.9698	0.9762	0.9836
Fold 3	0.9965	0.9979	0.9871	0.9754	0.9809	0.9867
Fold 4	0.9965	0.9980	0.9859	0.9768	0.9810	0.9874
Fold 5	0.9965	0.9979	0.9869	0.9736	0.9798	0.9858
Fold 6	0.9960	0.9977	0.9838	0.9737	0.9783	0.9857
Fold 7	0.9952	0.9973	0.9739	0.9700	0.9717	0.9836
Fold 8	0.9947	0.9969	0.9797	0.9600	0.9688	0.9785
Fold 9	0.9956	0.9974	0.9849	0.9720	0.9780	0.9847
Fold 10	0.9960	0.9977	0.9835	0.9731	0.9775	0.9854
Avg.	0.9959	0.9976	0.9833	0.9719	0.9770	0.9847

The achieved results show that the proposed CNN should be readjusted when the data dimensions growing between 15 and 20%. In particular, the comparison between the whole dataset (size 100%) and the dataset reduced by 20% (size 80%) shows a worsening of all classification metrics. For instance, the proposed CNN has respectively obtained a worsening of 3% in average precision, 7% in average sensibility, and 6% in average F-score.

In order to show the effectiveness of the use of the proposed representation method, the achieved results have been compared with the most notable ML-based approaches implemented in the WEKA [45] framework. More precisely, we used multi-layer perceptron (MLP), J48 trees (J48), and NB, to derive the classification metrics by considering a flattened version of the employed dataset that has been used to train and test the proposed CNN. Table 8 summarizes the comparison between the proposed CNN (Pr-CNN) and the employed ML-based methods.

Table 6 Multi-class confusion matrix related to 70/30 criteria

	Adrd	Boqx	FakeDoc	Fusob	G.Master	GW	Iconosys	Kmin	Lotoor	Mseg
Adrd	22	0	0	0	0	0	0	0	0	0
Boqx	0	55	0	0	0	0	0	0	0	0
FakeDoc	0	0	31	0	0	0	0	0	0	0
Fusob	0	0	0	60	0	0	0	0	0	0
G.Master	0	0	0	0	35	0	0	0	1	0
GW	0	0	0	0	0	23	0	0	0	0
Iconosys	0	0	0	0	0	0	34	0	1	0
Kmin	0	0	0	0	0	0	0	43	0	0
Lotoor	0	4	0	0	0	0	0	0	78	0
Mseg	0	0	0	0	0	0	0	0	0	68

Table 7 Performance metrics related to dataset updating process

Size (%)	Acc.	Spec.	Prec.	Sens.	F-score	AUC
100	0.9974	0.9982	0.9906	0.9895	0.9898	0.9937
95	0.9977	0.9987	0.9853	0.9894	0.9872	0.9940
90	0.9949	0.9970	0.9786	0.9667	0.9711	0.9819
85	0.9914	0.9949	0.9666	0.9419	0.9529	0.9684
80	0.9889	0.9936	0.9623	0.9116	0.9246	0.9526
75	0.9667	0.9790	0.9188	0.7907	0.8132	0.8849

Table 8 Comparison between the proposed CNN and ML-based methods

	Acc.	Spec.	Prec.	Sens.	F-score	AUC
Pr-CNN	0.9974	0.9982	0.9906	0.9895	0.9898	0.9937
J48	0.9670	0.9670	0.9670	0.9680	0.9670	0.9670
NB	0.9647	0.9650	0.9650	0.9670	0.9640	0.9660
MLP	0.8348	0.8350	0.8330	0.8350	0.8340	0.8350

The following comparison shows that the MLP classifier is not able to distinguish different application categories by considering Android permissions and their severity levels, while J48 trees and the NB classifier have achieved good results. More precisely, the proposed CNN has obtained up to a 3% improvement in average accuracy over J48 trees and the NB classifier, and up to a 16% over MLP classifier. Consequently, the proposed CNN can reduce the number of FPs and FNs, and then, better minimize the classification error respect to the most famous ML-based approaches.

Finally, we compared the proposed CNN with the ML and DL based state-of-art solutions. We considered RF results respectively achieved by A. Kumar et al. (Kum-RF) [25] and N. Xie et al. (Xie-RF) [47], LSTM neural network results achieved by R Vinayakumar et al. (Vi-LSTM) [42], and DNN results obtained by C. Li et al. (Li-DNN) [26].

Table 9 Comparison between the proposed CNN and state-of-art solutions

	Acc.	Spec.	Prec.	Sens.	F-score	AUC
Pr-CNN	0.9974	0.9982	0.9906	0.9895	0.9898	0.9937
Li-DNN	0.9925	0.9945	0.9961	0.9904	0.9933	0.9925
Xie-RF	0.9770	0.9992	0.9775	0.9775	0.9775	0.9884
Kum-RF	0.9100	0.9200	0.9000	0.9300	0.9147	0.9250
Vi-LSTM	0.8970	0.6280	0.9100	0.9600	0.9147	0.7690

Table 9 summarizes the comparison between the Pr-CNN and the state-of-art solutions.

First of all, the following comparison shows that the Vi-LSTM and Kum-RF solutions have achieved discrete results, and consequently, the proposed CNN has obtained up to 10% and 8% in average accuracy over both solutions, respectively. As reported in Sect. 2, Vi-LSTM evaluation metrics have been obtained by only considering Android permissions translated as numerical information, while Kum-RF evaluation metrics have been achieved by considering Grayscale images directly generated from the APK files, without performing any code extraction and decompiling operations. Consequently, the selected static features are not sufficient to achieve equivalent results as those obtained by the proposed CNN. Second, Xie-RF and Li-DNN have been achieved optimal results, and consequently, the proposed CNN has obtained up to 2% in average accuracy over Xie-RF, while their evaluation metrics are similar to those achieved by Li-DNN. However, the proposed Perm-Map representation technique is only based on Android permission and their severity levels, while Xie-RF and Li-DNN are based on Android permissions and Java methods. Consequently, Xie-RF and Li-DNN become ineffective against obfuscation techniques. Finally, Table 10 reports a final overview among proposed CNN, ML-based methods of WEKA, and state-of-art solutions.

Table 10 Overview among proposed CNN, ML-based methods of WEKA, and state-of-art solutions

	Acc.	Spec.	Prec.	Sens.	F-score	AUC
Pr-CNN	0.9974	0.9982	0.9906	0.9895	0.9898	0.9937
Li-DNN	0.9925	0.9945	0.9961	0.9904	0.9933	0.9925
Xie-RF	0.9770	0.9992	0.9775	0.9775	0.9775	0.9884
J48	0.9670	0.9670	0.9670	0.9680	0.9670	0.9670
NB	0.9647	0.9650	0.9650	0.9670	0.9640	0.9660
Kum-RF	0.9100	0.9200	0.9000	0.9300	0.9147	0.9250
Vi-LSTM	0.8970	0.6280	0.9100	0.9600	0.9147	0.7690
MLP	0.8348	0.8350	0.8330	0.8350	0.8340	0.8350

6.5 Feature selection process

Since the number of employed permissions is 298, the final goal is devoted to exploring a feature extraction technique, based on the most frequent Android permissions, in order to reduce the computational effort required by the generation and training processes of the Perm-Map and CNN, respectively. To this purpose, we have analyzed the permissions frequencies distribution in order to find the minimum frequency number that was able to reduce the number of employed permissions and preserve the number of applications analyzed previously. We have performed the following analysis by using a dedicated Python script. More precisely, we have firstly created an ordered dictionary to store each permission and its frequency. Then, we have considered all Android permissions required at least 50 times, and consequently, 57 Android permissions have been considered for the generation process of each Perm-Map. Figure 7 shows the first five most required Android permissions.

Subsequently, according to the workflow shown in Fig. 2, we employed the 57 Android permissions to generate and store each Perm-Maps as a matrix 4×64 in accordance with the maximum number of distinct severity levels and an over-bound number of Android permissions, respectively. We have chosen the following over-bound to simplify the operations that are performed by convolutional layers. Thus, we have split the following new dataset in order to run the experiments. To this purpose, the whole dataset has been subdivided into two mutually exclusive subsets assuming the role of learning and testing datasets, respectively. We used 70% of the entire dataset for learning and the remaining 30% for testing. The employed neural network has been compiled with Adam optimizer, SparseCategoricalFocalLoss function, batch_size = 64, and 150 epochs. Furthermore, it presents the same architecture of the neural network described in Fig. 6 except for the input_shape = (4, 64, 1) and dense layers with dropout = 0.45. Finally, the computational effort for the text substitution, Perm-Maps generation, and training processes have been derived with and without considering the employed features selection method, respectively. Table 11 reports the computational effort required for each analyzed phase, Table 12 shows results that have been obtained from the testing phase by using the 70/30 criteria, while Table 13 summarizes the comparison between the proposed CNNs that have been respectively called CNN-NoExtraction (CNN-NE) and CNN-WithExtraction (CNN-WE).

The obtained results show that the employed feature selection approach could reduce the computational effort required by each analyzed process. More precisely, Table 11 shows that text substitution and Perm-Maps generation processes have been slightly improved, respectively. Furthermore, it shows that the training process has been improved by 3.5 s, while the total effort has been improved by 3.6 s. Finally, the comparison reported in Table 13 demonstrates that proposed CNNs have been obtained equivalent evaluation metrics by testing phase, and thus, how the employed features selections criteria could also optimize the proposed representation approach.

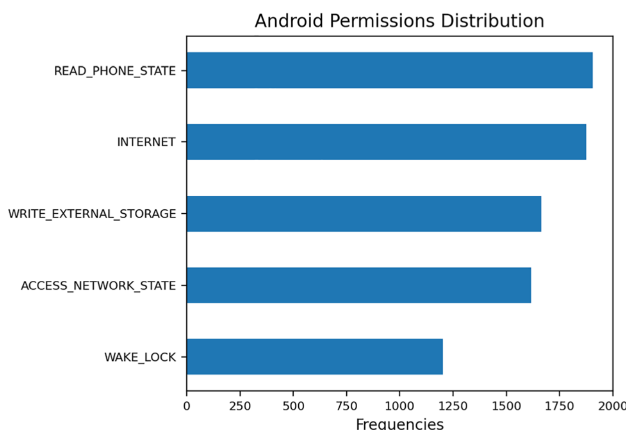


Fig. 7 Most required Android permissions

Table 11 Required computational effort

	No sel. (s)	With sel. (s)	Diff. (s)
Text sub.	0.255570	0.152351	0.103219
Perm-Maps gen.	0.058101	0.046116	0.011985
Training	11.589643	8.073685	3.515958
Total	11.903314	8.272152	3.631162

Table 12 Performance metrics derived by the 70/30 criteria and features selection method

	Acc.	Spec.	Prec.	Sens.	F-score	AUC
Adrd	0.9978	1.0000	1.0000	0.9500	0.9744	0.9750
Boqx	0.9956	1.0000	1.0000	0.9688	0.9841	0.9844
FakeDoc	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
Fusob	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
G.Master	0.9933	0.9976	0.9773	0.9556	0.9663	0.9766
GW	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
Iconosys	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
Kmin	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
Lotoor	0.9911	0.9894	0.9524	1.0000	0.9756	0.9947
Mseg	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
Avg.	0.9978	0.9987	0.9930	0.9874	0.9900	0.9931

Table 13 Comparison between the proposed CNNs

	Acc.	Spec.	Prec.	Sens.	F-score	AUC
CNN-NE	0.9974	0.9982	0.9906	0.9895	0.9898	0.9937
CNN-WE	0.9978	0.9987	0.9930	0.9874	0.9900	0.9931

7 Conclusions and future works

In this paper, novel features called Perm-Maps, based on Android permissions and their corresponding severity levels, have been presented. Next, a CNN has been used to show the potentialities of the proposed approach. More precisely, it has been enhanced by a training process based on a federated logic, where end-users devices extract static features locally and send them to a central server devoted to training a neural network performing malware classification. Then, the effectiveness of the presented methodology has been validated by using statistic metrics and comparing it to the most popular state-of-the-art ML-based approaches, like NB, MLP and J48 DTs. The obtained results show that the proposed CNN has achieved up to a 3% improvement in average accuracy over a J48 tree-based and NB classifier, and up to 16% over a MLP classifier, respectively. Finally, a feature selection technique, based on the most frequent Android permissions, has been explored to reduce the computational effort required by the Perm-Maps generation and CNN training processes, respectively. The achieved results show that the proposed methodology has improved the training time by 3.6 s and that they are also comparable with those obtained without considering any features selection technique.

However, due to the high number of existing Android-based applications, we would like to propose two possible future works. First of all, we will investigate the proposed features by considering an enormous quantity of decentralized data and applying a fully federated learning approach, involving end devices in model construction. Finally, since the most popular ML and DL based methods consider only features obtained at the end of malware analysis, we will propose new solutions capable of reducing damages caused at run-time by processing streams of dynamic features. For instance, several combinations among LSTM layers, CNNs, and stacked AEs (SAEs) could be explored and combined with the proposed approach.

Funding The author(s) received no financial support for the research, authorship, and/or publication of this article.

Data availability The used data is available on http://antlab.di.unisa.it/malware/.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

Informed consent Informed consent was obtained from all individual participants included in the study.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. Android: define a custom app permission. https://developer.android.com/guide/topics/permissions/defining. Accessed 2021
2. Android: Manifest.permission. https://developer.android.com/reference/android/Manifest.permission. Accessed 2021
3. Android: $\langle {{\rm permission}}\rangle $. https://developer.android.com/guide/topics/manifest/permission-element. Accessed 2021
4. Android: permissions on Android. https://developer.android.com/guide/topics/permissions/overview. Accessed 2021
5. Android: R.attr | protectionlevel. https://developer.android.com/reference/android/R.attr. Accessed 2021

6. Aonzo, S., Georgiu, G.C., Verderame, L., Merlo, A.: Obfuscapk: an open-source black-box obfuscation tool for Android apps. SoftwareX **11**, 100403 (2020). https://doi.org/10.1016/j.softx.2020.100403
7. Aonzo, S., Merlo, A., Migliardi, M., Oneto, L., Palmieri, F.: Low-resource footprint, data-driven malware detection on Android. IEEE Trans. Sustain. Comput. **5**(2), 213–222 (2020)
8. Arp, D., Spreitzenbarth, M., Hübner, M., Gascon, H., Rieck, K.: Drebin: effective and explainable detection of Android malware in your pocket (2014). https://doi.org/10.14722/ndss.2014.23247
9. Bhagwat, R., Abdolahnejad, M., Moocarme, M.: Applied Deep Learning with Keras: Solve Complex Real-Life Problems with the Simplicity of Keras. Packt Publishing, Birmingham (2019)
10. Carrez, F., Elsaleh, T., Gómez, D., Sánchez, L., Lanza, J., Grace, P.: A reference architecture for federating IoT infrastructures supporting semantic interoperability. In: 2017 European Conference on Networks and Communications (EuCNC), pp. 1–6 (2017). https://doi.org/10.1109/EuCNC.2017.7980765
11. CuckooDroid: CuckooDroid Book. https://cuckoo-droid.readthedocs.io/en/latest/. Accessed 2020
12. D'Angelo, G., Palmieri, F., Robustelli, A., Castiglione, A.: Effective classification of Android malware families through dynamic features and neural networks. Connect. Sci. (2021). https://doi.org/10.1080/09540091.2021.1889977
13. D'Angelo, G., Ficco, M., Palmieri, F.: Malware detection in mobile environments based on autoencoders and API-images. J. Parallel Distrib. Comput. **137**, 26–33 (2020). https://doi.org/10.1016/j.jpdc.2019.11.001
14. David, O., Netanyahu, N.S.: DeepSign: deep learning for automatic malware signature generation and classification. In: International Joint Conference on Neural Networks (IJCNN) pp. 1–8 (2015)
15. Desnos, A., Gueguen, G.: Androguard. https://github.com/androguard/androguard. Accessed 2020
16. Ficco, M.: Detecting IoT malware by Markov chain behavioral models. pp. 229–234 (2019). https://doi.org/10.1109/IC2E.2019.00037
17. G DATA: G DATA Mobile Malware Report 2019: New High for Malicious Android Apps. https://www.gdatasoftware.com/news/g-data-mobile-malware-report-2019-new-high-for-malicious-android-apps. Accessed 2020
18. Guardsquare: DexGuard. Guardsquare. https://www.guardsquare.com/dexguard. Accessed 2021
19. Guardsquare: ProGuard. Guardsquare. https://www.guardsquare.com/proguard. Accessed 2021
20. Idanr: CuckooDroid—automated Android malware analysis. https://github.com/idanr1986/cuckoo-droid. Accessed 2020
21. Karbab, E.B., Debbabi, M., Derhab, A., Mouheb, D.: MalDozer: automatic framework for Android malware detection using deep learning. Digit. Investig. **24**, S48–S59 (2018). https://doi.org/10.1016/j.diin.2018.01.007
22. Kaspersky: Android mobile security threats. https://www.kaspersky.com/resource-center/threats/mobile. Accessed 2020
23. Kelaidonis, D., Rouskas, A., Stavroulaki, V., Demestichas, P., Vlacheas, P.: A federated edge cloud-IoT architecture. In: 2016 European Conference on Networks and Communications (EuCNC), pp. 230–234 (2016). https://doi.org/10.1109/EuCNC.2016.7561038
24. Kolosnjaji, B., Zarras, A., Webster, G., Eckert, C.: Deep learning for classification of malware system call sequences. In: Kang, B.H., Bai, Q. (eds.) AI 2016: Advances in Artificial Intelligence, pp. 137–149. Springer, Cham (2016)
25. Kumar, A., Sagar, K.P., Kuppusamy, K.S., Aghila, G.: Machine learning based malware classification for Android applications using multimodal image representations. In: 2016 10th International Conference on Intelligent Systems and Control (ISCO), pp. 1–6 (2016). https://doi.org/10.1109/ISCO.2016.7726949
26. Li, C., Mills, K., Niu, D., Zhu, R., Zhang, H., Kinawi, H.: Android malware detection based on factorization machine. IEEE Access **7**, 184008–184019 (2019). https://doi.org/10.1109/ACCESS.2019.2958927
27. Li, D., Wang, Z., Xue, Y.: Fine-grained Android malware detection based on deep learning. In: 2018 IEEE Conference on Communications and Network Security (CNS), pp. 1–2 (2018). https://doi.org/10.1109/CNS.2018.8433204
28. Li, Y., Jang, J., Hu, X., Ou, X.: Android malware clustering through malicious payload mining. In: Lecture Notes in Computer Science, pp. 192–214 (2017). http://arxiv.org/abs/1707.04795
29. Lin, T.Y., Goyal, P., Girshick, R., He, K., Dollár, P.: Focal loss for dense object detection. IEEE Trans. Pattern Anal. Mach. Intell. **42**(2), 318–327 (2018)
30. Loria, M.P., Toja, M., Carchiolo, V., Malgeri, M.: An efficient real-time architecture for collecting IoT data. In: 2017 Federated Conference on Computer Science and Information Systems (FedCSIS), pp. 1157–1166 (2017). https://doi.org/10.15439/2017F381
31. Martín García, A., Rodriguez-Fernandez, V., Camacho, D.: CANDYMAN: classifying Android malware families by modelling dynamic traces with Markov chains. Eng. Appl. Artif. Intell. (2018). https://doi.org/10.1016/j.engappai.2018.06.006
32. McAfee: McAfee mobile threat report. https://www.mcafee.com/content/dam/consumer/en-us/docs/2020-Mobile-Threat-Report.pdf. Accessed 2020
33. McLaughlin, N., Martinez del Rincon, J., Kang, B., Yerima, S., Miller, P., Sezer, S., Safaei, Y., Trickel, E., Zhao, Z., Doupé, A., Joon Ahn, G.: Deep Android malware detection. In: Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, CODASPY '17, pp. 301–308. Association for Computing Machinery, New York (2017). https://doi.org/10.1145/3029806.3029823
34. Onwuzurike, L., Mariconti, E., Andriotis, P., Cristofaro, E.D., Ross, G., Stringhini, G.: MaMaDroid: detecting Android malware by building Markov chains of behavioral models (extended version). ACM Trans. Priv. Secur. (2019). https://doi.org/10.1145/3313391
35. Prabhu, T.N.: Exploratory data analysis in Python. https://towardsdatascience.com/exploratory-data-analysis-in-python-c9a77dfa39ce. Accessed 2020
36. Sadat, M.N., Al Aziz, M.M., Mohammed, N., Chen, F., Jiang, X., Wang, S.: SAFETY: Secure gwAs in Federated Environment through a hYbrid Solution. IEEE/ACM Trans. Comput. Biol. Bioinform. **16**(1), 93–102 (2019). https://doi.org/10.1109/TCBB.2018.2829760
37. Sanyal, S., Wu, D., Nour, B.: A federated filtering framework for internet of medical things. In: ICC 2019—2019 IEEE International Conference on Communications (ICC), pp. 1–6 (2019). https://doi.org/10.1109/ICC.2019.8761381
38. Shih, C., Chuang, C., Yeh, H.: Federating public and private intelligent services for IoT applications. In: 2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC), pp. 558–563 (2017). https://doi.org/10.1109/IWCMC.2017.7986346
39. Spreitzenbarth, M., Freiling, F., Echtler, F., Schreck, T., Hoffmann, J.: Mobile-sandbox: having a deeper look into Android applications. In: Proceedings of the 28th Annual ACM Symposium on Applied Computing, SAC '13, pp. 1808–1815. Association for Computing Machinery, New York (2013). https://doi.org/10.1145/2480362.2480701

40. Sruthi, V.M., Thanudas, B., Sreelal, S., Chakraborty, A., Manoj, B.S.: ACTM: API call transition matrix-based malware detection method. In: 2018 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS), pp. 1–6 (2018). https://doi.org/10.1109/ANTS.2018.8710081
41. TechCrunch: 6.1B smartphone users globally by 2020, overtaking basic fixed phone subscriptions. https://techcrunch.com/2015/06/02/6-1b-smartphone-users-globally-by-2020-overtaking-basic-fixed-phone-subscriptions. Accessed 2020
42. Vinayakumar, R., Soman, K.P., Poornachandran, P.: Deep Android malware detection and classification. In: 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI), pp. 1677–1683 (2017). https://doi.org/10.1109/ICACCI.2017.8126084
43. Wang, H., Li, X., Ji, H., Zhang, H.: Federated offloading scheme to minimize latency in MEC-enabled vehicular networks. In: 2018 IEEE Globecom Workshops (GC Wkshps), pp. 1–6 (2018). https://doi.org/10.1109/GLOCOMW.2018.8644315
44. Wei, F., li, Y., Roy, S., Ou, X., Zhou, W.: Deep ground truth analysis of current Android malware. pp. 252–276 (2017). https://doi.org/10.1007/978-3-319-60876-1_12
45. WEKA: WEKA 3—data mining with open source machine learning software in Java. https://www.cs.waikato.ac.nz/ml/weka/. Accessed 2020
46. Weng, J.: Exploratory data analysis: a practical guide and template for structured data. https://towardsdatascience.com/exploratory-data-analysis-eda-a-practical-guide-and-template-for-structured-data-abfbf3ee3bd9. Accessed 2020
47. Xie, N., Zeng, F., Qin, X., Zhang, Y., Zhou, M., Lv, C.: RepassDroid: automatic detection of Android malware based on essential permissions and semantic features of sensitive APIs. In: 2018 International Symposium on Theoretical Aspects of Software Engineering (TASE), pp. 52–59 (2018). https://doi.org/10.1109/TASE.2018.00015
48. Yang, S.: An image-inspired and CNN-based Android malware detection approach. In: Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering, ASE '19, pp. 1259–1261. IEEE Press (2019). https://doi.org/10.1109/ASE.2019.00155

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Gianni D'Angelo received the M.S. Degree (cum laude) in Computer Engineering, and the Ph.D. Degree in Computer Science, Applied Electromagnetism and Telecommunications from the University of Salerno, Salerno, Italy, in 1998 and 2003, respectively. Since 2012, he is Contract Professor of "Elements of Computer Science" at the Department of Law, Economics, Management and Quantitative Methods (DEMM) of the University of Sannio, Benevento, Italy. He also is Contract Professor of the following courses at the University of Salerno, Italy: "Fundamentals of Computer Science and Programming" at the Department of Industrial Engineering, and "Computer Networks" at the Department of Computer Science. He also serves in the same University as Research Fellow at the Department of Computer Science. His research interests concern with the development and implementation of soft computing algorithms for high-performance machines—HPC, and parallel computing for knowledge discovery in Big Data context. He gained experience in the world of the pattern recognition, neural networks, fuzzy logic, ANFIS systems, genetic and evolutionary algorithms, and parallel programming applied in various scientific and industrial fields. He authored many articles published in international journals, books and conferences, and currently serves as a reviewer, editorial board and guest editor for several international journals.

Francesco Palmieri is a Full Professor at the University of Salerno, Italy. He received from the same university an Italian M.S. "Laurea" Degree and a Ph.D. in Computer Science. His major research interests concern high performance networking protocols and architectures, routing algorithms and network security. Previously he has been an Assistant Professor at the Second University of Naples, and the Director of the Telecommunication and Networking Division of the Federico II University, in Naples, Italy. At the start of his career, he also worked for several international companies on networking-related projects. He has been closely involved with the development of the Internet in Italy as a Senior Member of the Technical-Scientific Advisory Committee and of the CSIRT of the Italian NREN GARR. He has published a large number of papers (more than 200) in leading technical journals, books and conferences and currently serves as the editor-in-chief of an international journal and is part of the editorial board or associate editor of several other well reputed ones.

Antonio Robustelli Is a Ph.D. Student at the University of Salerno, Italy. He received from the same university the M.S. Degree (cum laude) in Computer Science. His research interests concern the application of AI-based solution to face security issues related to IoT domains and ICT infrastructures.