International Journal of Information Security (2022) 21:725–738
https://doi.org/10.1007/s10207-022-00579-6

REGULAR CONTRIBUTION

MAPAS: a practical deep learning-based android malware detection system

Jinsung Kim[1] · Younghoon Ban[1] · Eunbyeol Ko[2] · Haehyun Cho[2] · Jeong Hyun Yi[2]

Published online: 9 February 2022

Abstract
A lot of malicious applications appears every day, threatening numerous users. Therefore, a surge of studies have been conducted to protect users from newly emerging malware by using machine learning algorithms. Albeit existing machine or deep learning-based Android malware detection approaches achieve high accuracy by using a combination of multiple features, it is not possible to employ them on our mobile devices due to the high cost for using them. In this paper, we propose MAPAS, a malware detection system, that achieves high accuracy and adaptable usages of computing resources. MAPAS analyzes behaviors of malicious applications based on API call graphs of them by using convolution neural networks (CNN). However, MAPAS does not use a classifier model generated by CNN, it only utilizes CNN for discovering common features of API call graphs of malware. For efficiently detecting malware, MAPAS employs a lightweight classifier that calculates a similarity between API call graphs used for malicious activities and API call graphs of applications that are going to be classified. To demonstrate the effectiveness and efficiency of MAPAS, we implement a prototype and thoroughly evaluate it. And, we compare MAPAS with a state-of-the-art Android malware detection approach, MaMaDroid. Our evaluation results demonstrate that MAPAS can classify applications 145.8% faster and uses memory around ten times lower than MaMaDroid. Also, MAPAS achieves higher accuracy (91.27%) than MaMaDroid (84.99%) for detecting unknown malware. In addition, MAPAS can generally detect any type of malware with high accuracy.

Keywords Malware detection · Deep learning · Deep learning interpretation · API Call graph analysis

1 Introduction

In the fourth quarter of 2019, 35 millions of malware targeting mobile devices appeared [1]. On average, about 15 malicious applications appeared per minute. Due to the threat, many commercial antivirus products such as Bitdefender, Norton, McAfee, BullGuard, Panda, Kaspersky, ESET, Avira, Avast were launched. However, their critical limitation is that they cannot detect unknown malware because they rely on signatures of known malicious applications [66]. Therefore, the research community have been focusing on developing malware detection approaches by using a machine learning or deep learning algorithm with various features for protecting users from emerging malware [2,5,7,10,12,14–16,18–23,26,27,30–34,36–42,45–51,54,58,61–64,66,67,70,72–74,76–78,81–90,92]. In particular, a lot of malware detection approach using deep learning algorithms were recently introduced [23,30,31,34,38,45,47,49,51,54,58,77,78,85,88,92].

However, previous deep learning-based malware detection approaches commonly require very high *cost* (in terms of computing resources) for using them because they use a combination of multiple features to achieve the high accuracy [71]. For example, a classifier model generated by the convolutional neural network (CNN) requires enormous amount of memory for classifying data [44]. Consequently, albeit

✉ Jeong Hyun Yi
jhyi@ssu.ac.kr

Jinsung Kim
okokabv@soongsil.ac.kr

Younghoon Ban
byhoon6279@soongsil.ac.kr

Eunbyeol Ko
kongstar159@soongsil.ac.kr

Haehyun Cho
haehyun@ssu.ac.kr

[1] School of Software Convergence, Soongsil University, Seoul 06978, Korea

[2] School of Software, Soongsil University, Seoul 06978, Korea

previously proposed deep learning-based malware detection systems could achieve very high accuracy, it is unlikely to employ them on our mobile devices of which computing resources are limited or personal computers. Therefore, it is of great importance to develop a malware detection approach that can protect users from newly emerging malware *and* can be practically used.

In this work, we propose a practical malware detection system, MAPAS, that achieves high accuracy against known and unknown malware as well as adaptable usages of computing resources. MAPAS learns behaviors of malicious applications based on API call graphs by using a deep learning algorithm (CNN). Then, it detects malware based on common patterns of API call graphs of malware. For efficiently detecting malware, MAPAS does not utilize a classifier model created by CNN but uses a lightweight classifier that calculates a similarity score between API call graphs used for malicious activities and API call graphs of applications that are going to be classified by using the Jaccard Similarity algorithm [3].

To show the effectiveness and efficiency of MAPAS, we thoroughly evaluate our prototype and compare it with a state-of-the-art Android malware detection approach, MaMaDroid [61]. MaMaDroid also utilizes API call graphs for detecting malware based on their behaviors. Our evaluation results demonstrate that MAPAS achieves better performance in terms of a processing time to classify applications and MAPAS uses much lower memory than the previous approach. Specifically, MAPAS classifies applications 145.8% faster and uses memory around ten times lower than MaMaDroid (when it used the random forest algorithm). In addition, MAPAS achieves higher accuracy (91.27%) than MaMaDroid (84.99%) for detecting unknown malware (i.e., when they classify newer malware released later than ones in our training dataset).

In summary, this paper makes the following contributions:

- We propose a practical Android malware detection system, MAPAS, that find malware based on malicious behavioral features. To this end, MAPAS learns API call graphs of malware and detects malware based on analyzed patterns of API call graphs used for malicious behaviors. MAPAS employs a deep learning algorithm not to use a classifier model generated by the algorithm but only to discover common features of malware. MAPAS performs malware detection with a lightweight classifier for the efficiency.
- We implement a prototype of MAPAS and thoroughly evaluate it. Also, we compare MAPAS against MaMaDroid to demonstrate the effectiveness and efficiency of it. Our evaluation results show that MAPAS achieves better performance than MaMaDroid in terms of the usage of computing resources as well as the accuracy for detecting new malware. Also, MAPAS can generally detect any type of malware with high accuracy.

This paper is organized as follows. We first provide technical backgrounds in Sect. 2. Section 3 explains the goals of MAPAS and presents the specific design approach in Section 4. We evaluate MAPAS to demonstrate its effectiveness and efficiency in Sect. 5. Previous studies are discussed in Sect. 6. Finally, Sect. 7 discusses the conclusion.

We release the source code of our proof-of-concept implementation at https://github.com/okokabv/MAPAS.

2 Background

In this section, we introduce malware detection methods and a common limitation of machine/deep learning-based Android malware detection approaches, the mainstream of malware detection approaches, that hinders practical uses of them.

2.1 Detecting android malware

Android malware detection approaches can be categorized into two groups based on analysis methods (i.e., dynamic analysis and static analysis) used to collect features of malware: (1) dynamic analysis-based malware detection approaches and (2) static analysis-based ones.

Dynamic analysis-based malware detection approaches have an advantage over static analysis-based approaches in analyzing concrete behaviors of malware [5,12,14,22,26,27,32,36,67,70,73,74,81,83,87,90]. Also, they have another advantage of analyzing malware equipped with anti-analysis mechanisms such as obfuscation. However, typically the dynamic analysis method consumes a lot of resources and time because we actually need to execute applications.

On the other hand, static analysis-based malware detection approaches identify features of malware without executing them, and thus, the cost for analyzing each application is much lower than dynamic analysis-based approaches in general [2,7,10,15,16,18–21,23,30,31,34,38,39,42,45–47,49–51,54,58,63,64,66,72,76–78,82,84–86,88,89,92]. Because of the advantage of using less computing resources and high accuracy in static analysis-based malware detection approaches, most malware detection approaches employ the static analysis method for extracting malware's features.

2.2 Typical features used for static analysis-based malware detection approaches

The first step to develop a malware detection system is to decide features of malware to distinguish them from benign applications. Typically, developer-written descriptions, user

reviews, permissions, opcode and APIs are used as such features.

Developer-written descriptions A couple of research work employed developer-written descriptions on applications as a key feature for detecting malware [53,62]. However, detecting malware based on developer-written descriptions is not reliable because inferring accurate execution behaviors of applications is unlikely possible.

User reviews Among Android malware detection approaches, there were attempts that employ user reviews as an important feature [33,41]. However, similar to the malware detection approaches that use developer-written descriptions, the accuracy is not high enough to be used in a practical manner because user reviews usually do not contain concrete explanations on applications that can be used for detecting malware.

Opcode Several previous work showed there are common patterns of opcode that can be used to classify malicious applications [16,54,66,85]. They used common patterns of opcode such as *move* and *invoke* of bytecode in malicious applications.

Permissions There have been many research work for detecting malware based on permissions that applications require (e.g., a user's location, phone information, a mobile device's network status etc.) [10,19,23,42,46,63,64,76]. These approaches detect malware by using commonly used permissions such as network permission with users' location in malicious applications. However, Avdiienko et al. [11] showed that similar to malware, most benign Android applications access sensitive information of users and use a lot of permissions that are also typically used in malware. Consequently, permission-based malware detection approaches could incur a high false positive rate.

APIs Many approaches attempted to classify malicious applications based on APIs used in them [2,18,30,34,37,40,58,61]. By analyzing APIs used in an applications, we can understand functionalities that the application provide to users. For example, if an application uses APIs such as *android.telephony* and *android.telecom*, we can know that the application would monitor a mobile phone's network status and manages phone calls. As such, Android APIs provides functional information about what an application does. Therefore, we can infer an application's behavior by using APIs used in the application. However, if we only use APIs as a key feature for identifying malware, we can have high false positives because analyzing APIs does not provide an application's concrete behaviors and there are a lot of common APIs used in both benign and malicious applications [11].

2.3 Unpractical machine/deep learning-based android malware detection approaches

Within several years, a surge of studies were proposed to detect Android malware by employing machine or deep learning-based approaches, which classified malicious application based on features discussed in the previous section (Sect. 2.2) [2,5,7,10,12,14–16,18–23,26,27,30–34,36–42,45–51,54,58,61–64,66,67,70,72–74,76–78,81–90,92]. Among them, recently proposed approaches usually employed deep learning algorithms which utilize artificial neural networks [23,30,31,34,38,45,47,49,51,54,58,77,78,85,88,92]. The notable advantage of deep learning algorithms is that they can eliminate the need of domain expertise and manual feature extraction because they learn features of data algorithmically [68]. However, previous approaches commonly require very high *cost* (in terms of computing resources and times) for using their approaches because they use a combination of multiple features to achieve the high accuracy [71]. Consequently, even though they could achieve the high accuracy, it is difficult to employ them in a practical manner due to the high cost for using them.

3 Goal

In this work, our goal is to detect malicious applications efficiently while achieving the high accuracy (1) to reduce the cost for detecting them and (2) to deal with the increasing Android malware. To this end, we optimize the Android malware detection process by using a deep learning algorithm with a deep learning interpretation approach for extracting dominant, common features used in malware.

Deep learning-based malware detection approaches showed the high accuracy but have the disadvantage of using a lot of computing resources and times (as discussed in Sect. 2.3). In general, the cost for using a deep learning algorithm (to construct a classifier model) and even for using the model to actually classify malware is very expensive because they used complex features for increasing the accuracy. In this paper, we use a deep learning algorithm with a deep learning interpretation approach not for classifying malicious applications from benign applications, but only for identifying high-weight features of malware. We, then, build a low-cost classifier that finds malicious applications based on only such high-weight features identified by a deep learning algorithm. In this way, we can avoid heuristic feature selection for detecting malware as well as we can reduce the usage of computing resources and times for detecting malware (Fig. 1).

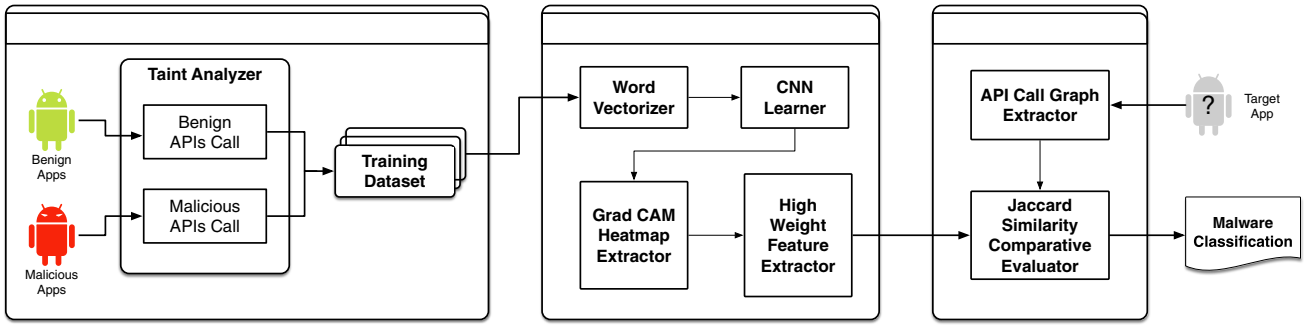

Fig. 1 Overview of MAPAS

4 Design

In this section, we first overview the proposed system, code-named MAPAS, (in Sect. 4.1) and demonstrate details of each step for detecting malware in Sects. 4.2, 4.3 and 4.4.

4.1 Design overview

Malware features used In this work, we attempts to detect malicious applications based on common patterns of their API call graphs. With API call graphs, we can find concrete malicious behaviors of malicious applications [20,50,72]. To be specific, MAPAS analyzes frequently used patterns of API call graphs which can lead to leakages of sensitive information (social security numbers, credit card numbers, passwords, etc.) with a deep learning algorithm. MAPAS, then, detects malware based on the identified patterns of malicious API call graphs.

The design of MAPAS consists of the following three steps:

(1) *Data Preprocessing* As the first step, MAPAS generates training dataset through extracting API call graphs from malicious and benign applications. Specifically, MAPAS obtains API call graphs by conducting the taint analysis with Flowdroid [9].
(2) *Identifying High-weight API Call Graphs* In this step, MAPAS first vectorizes training dataset and performs deep learning on the dataset by using convolution neural networks (CNN). After the learning phase finishes, MAPAS uses the deep learning interpretation approach, Grad-CAM, to discover high-weight API call graphs used in malicious applications.
(3) *Malware Detection* In the last step, MAPAS classifies malware by using the Jaccard algorithm which calculate the similarity between API call graphs of an application and the high-weight API call graphs of malicious applications.

4.2 Data preprocessing for generating training dataset

MAPAS extracts API call graphs of applications by conducting taint analysis. Taint analysis is a static analysis method used to track data flows in an application. Specifically, we use a taint analysis for analyzing data flows from specific *sources* that read sensitive data (e.g., a function reading a password) to *sinks* which can transfer data (e.g., a function writing to a socket) by identifying whether sensitive information can be leaked or not. Hence, we can find potential sensitive leakages from an application.

For MAPAS, we chose a static analysis tool based on evaluation results from Arzt [8] and Qiu et al. [65]. There are many taint analysis tools such as Flowdroid [9], AppScan [28], Epicc [60], JoDroid [56], DroidSafe [25] and Amandroid [80]. Among them, Arzt [8] and Qiu et al. [65] showed that overall Flowdroid has the best results in terms of the accuracy and the runtime performance. Therefore, in this work, we generates API call graphs based on taint analysis results from Flowdroid [9]. The detail process for generating API call graphs with Flowdroid is shown in Fig. 2.

It is worth noting that we exclude applications that have obfuscated API calls for the taint analysis. MAPAS uses Flowdroid that cannot extract API call graphs for API hiding techniques and class encryption techniques among obfuscation techniques such as renaming, control flow, string encryption, API hiding and class encryption [52]. Therefore, MAPAS has to exclude obfuscated applications that cannot extract API call graphs from Flowdroid. We leave this limitation as a future work (Fig. 3).

4.3 Deep learning and identifying high-weight API call graphs from malware

MAPAS uses a deep learning algorithm (CNN) [44] for the training dataset. While learning the dataset, the algorithm finds important features from the collected API call graphs used in malware and constructs the classification model. MAPAS, then, discovers the important features by using

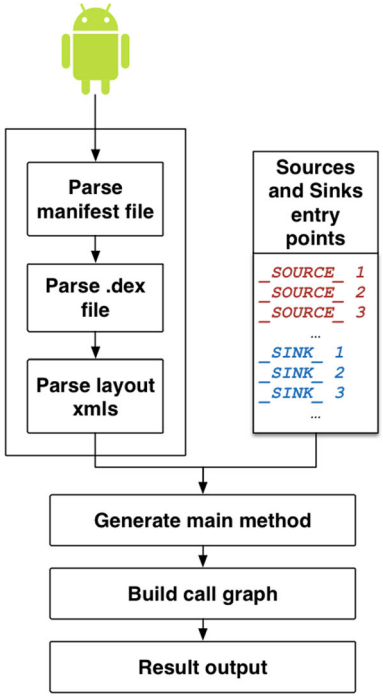

Fig. 2 Process of extracting API call graphs

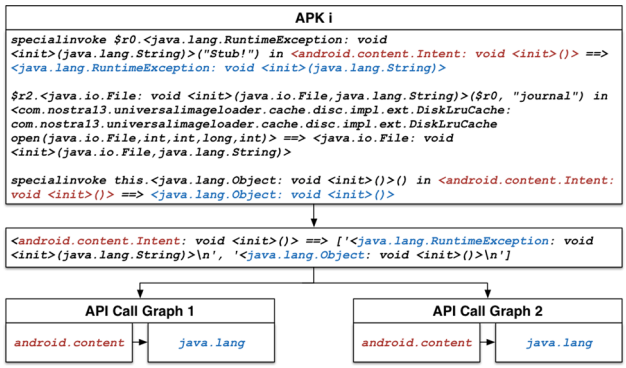

Fig. 3 Example of API call graphs extracted from an APK

a deep learning interpretation approach, Grad-CAM [69]. These features will directly be used to detect malicious applications with the Jaccard algorithm. (MAPAS does not use the classifier model generated by CNN.)

Vectorizing API Call Graphs In order to apply deep learning on API call graphs, which is text-type data, they must be converted into a vector. To vectorize text-type data, we can map each word in the data to an integer and create a vector with mapped integer numbers. Also, we can vectorize text-type data by analyzing the correlation between words known as word2vec [55] and analyzing the correlation between documents known as doc2vec [43]. MAPAS does not use vectorization methods such as word2vec and doc2vec but vectorizes API call graphs by simply mapping each API call graph to an integer number. For detecting malicious applications, API call graphs that MAPAS needs to find are specific sequences of function calls from the *sources* to the *sinks* as we discussed in Sect. 4.2. Each of malicious API call graphs represents a possible case of the sensitive information leak. Therefore, to detect malware, MAPAS should focus on finding the existence of such API call graphs rather than analyzing relationships between API call graphs.

Learning the dataset: MAPAS analyzes API call graphs commonly used in malware which can leak the sensitive information. To this end, MAPAS uses CNN [44] for learning the vectorized dataset. CNN is an effective deep learning algorithm for text-type data by using regional information of the data [35]. Please refer to "Appendix A" for the details on CNN. By learning the vectorized dataset with CNN, MAPAS can find common patterns of API call graphs that are frequently used in actual malicious applications. The overall learning process in MAPAS is illustrated in Fig. 4.

Finding high-weight features with a deep learning interpretation approach Deep learning models are a black-box model. Due to their multilayer and nonlinear structures, their predictions are not transparent [57]. CNN, also, operates in a black-box way, we cannot transparently figure out which API call graphs have *high weights* (which API call graphs are important) to detect malware from a classifier model generated by CNN. Hence, several deep learning interpretation approaches were proposed to transparently show specific data that substantially contributed to constructing a classifier model generated by a deep learning algorithm [4,29].

To observe high-weight API call graphs analyzed by CNN, MAPAS employs Grad-CAM [69] that produces visual explanations from CNN-based models. Please refer to "Appendix B" for more details on the approach.

As a result of using Grad-CAM, MAPAS found a high-weight API call graph of which the source is `android.content` and the sink is `java.net`. This call graph can leak user's sensitive information over the network.

After discovering high-weight features with Grad-CAM, MAPAS can classify malicious applications from benign ones based on such features. Note that MAPAS does not detect malware with the classifier model generated by CNN for reducing the *cost* in terms of the usage of computing resources. In Sect. 5, we demonstrate the effectiveness and efficiency of MAPAS by comparing it to the classifier model generated by CNN (Fig. 5).

4.4 Malware detection

For detecting malicious applications, MAPAS measures the similarity between two sets (the high-weight API call graphs and call graphs extracted from an unclassified application) by using Jaccard similarity algorithm [3] as shown in Fig. 6.

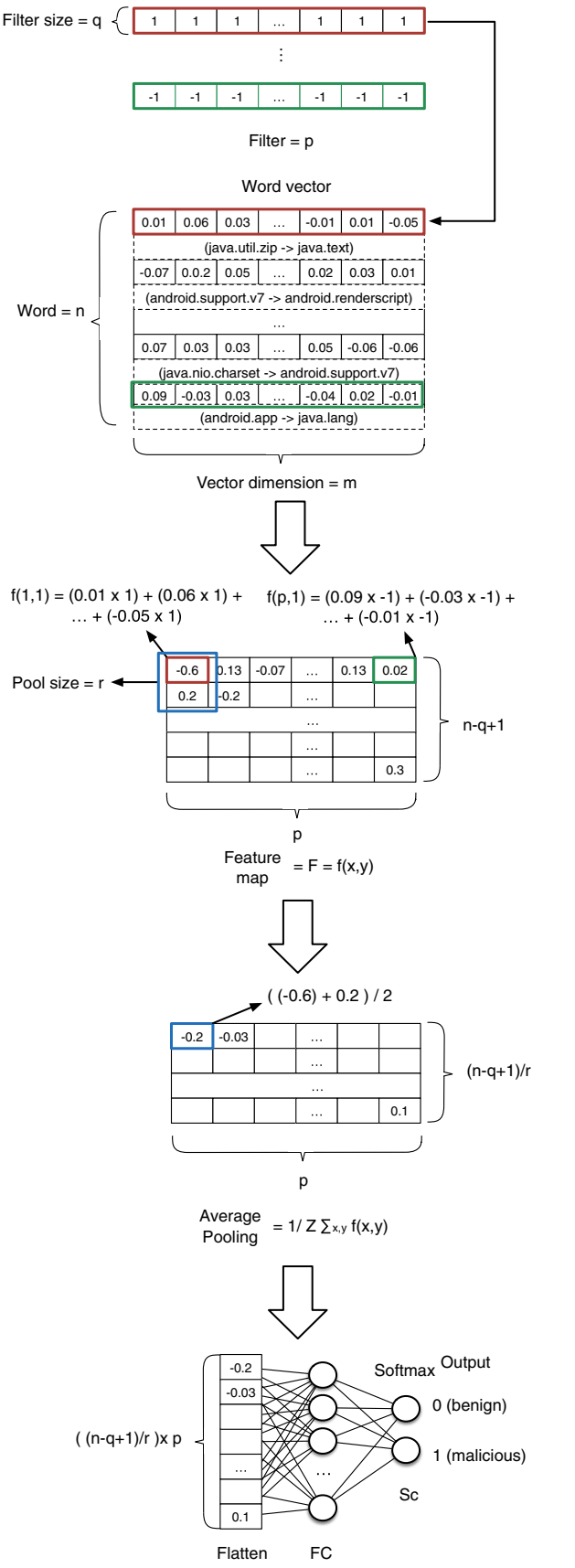

Fig. 4 Learning process using vectorized API call graphs with CNN

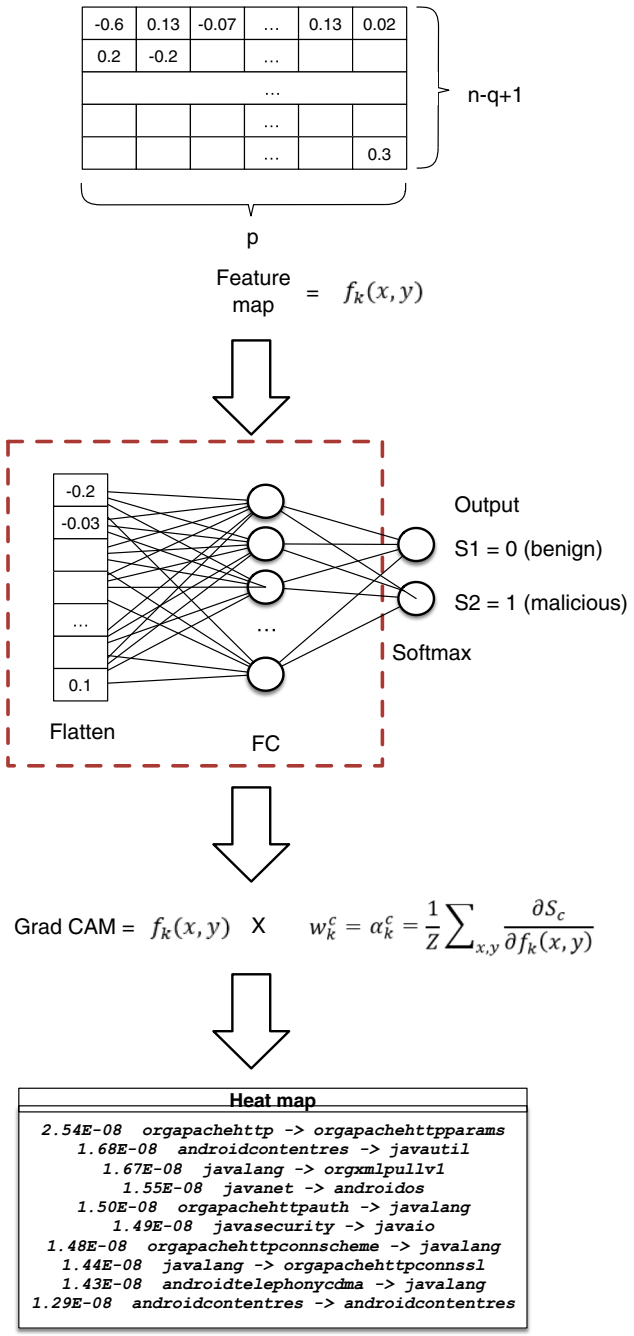

Fig. 5 A process for finding high-weight features using Grad-CAM

The Jaccard similarity has a value between 0 and 1. If two sets are exactly equal to each other, the similarity score is 1, and if two sets are totally different, the similarity score is 0. The expression of Jaccard similarity algorithm is as follows.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \quad (1)$$

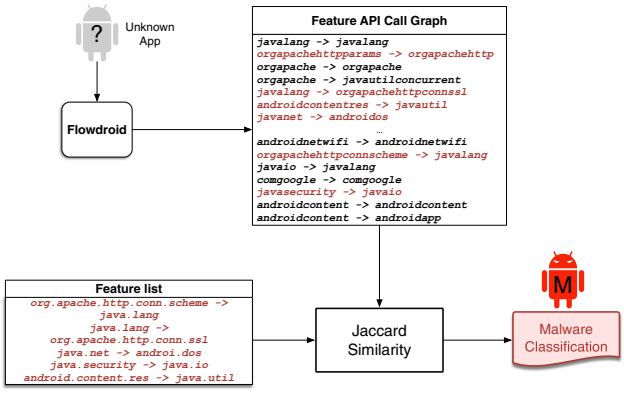

Fig. 6 Malware classification process of MAPAS

MAPAS considers an application is malware if the similarity score is higher than a threshold (0.4303) that we set based on testing results as in Sect. 5.2.

5 Evaluation

In this section, we evaluate MAPAS to demonstrate its effectiveness and efficiency. Our evaluation addresses the following research questions:

RQ 1. How much computing resources does MAPAS use to detect malware?
In Sect. 5.3, we evaluate the efficiency of the MAPAS's malware detection process by comparing it with the efficiency of a classifier model that is generated by a deep learning algorithm. In Sect. 5.4, we also compare the efficiency of MaMaDroid [61] with MAPAS.

RQ 2. How accurately can MAPAS detect malware?
In Sect. 5.4, we first evaluate the effectiveness of MAPAS by measuring the accuracy of malware detection results.

RQ 3. Can MAPAS detect newly emerging malicious applications?
We evaluate the effectiveness of MAPAS against malicious applications created later than the training datasets in Sect. 5.4.

5.1 Experimental configuration

Setup We performed our evaluations on a workstation running Ubuntu 18.04 with a 20-core Intel Xeon Gold 6230 CPU at 2.10 GHz, 128 GB RAM and a NVIDIA GeForce RTX 2080 GPU.

Datasets We first collected the top 10,000 applications from Google Play Store [24]. We, then, randomly downloaded 10,653 malicious applications released in 2018 and 2019 from VirusShare [75]. In addition, we used 23,039 malicious applications from Android Malware Dataset (AMD) [79]. Wei et al. classified the AMD into 70 categories [79].

Table 1 shows the number of applications used for our evaluation. Training dataset is used for generating a classifier model with CNN. We used Test dataset for evaluating the effectiveness of MAPAS.

Hyper-parameters In order to minimize the usage of computing resources in the learning phase, MAPAS uses one layer of the convolution layer and one layer of the pooling layer. Specifically, MAPAS uses the following hyper-parameters: Embedding layer: 64 dimensions; Convolution layer: *filters* = 32, *kernel_size* = 1, and the rest use default values; Pooling layer: max pooling; Compile: *optimizer* = '*rmsprop*', *loss* = '*binary_crossentropy*', *batch_size* = 500, *epochs* = 100. The total number of nodes used in the CNN model is 1,128,089.

5.2 Finding high-weight features

Training dataset 9000 malicious applications provided by VirusShare [75] and 9,000 benign applications downloaded from Google Play Store [6] were used for training a classifier model with CNN. To this end, we extracted API call graphs from the 18,000 applications by using Flowdroid [9]. In total, we obtained 21,690 unique API call graphs and used them as a training dataset.

Model learning and verification We trained a classifier model by using CNN with the training dataset. Next, we verified the classifier model by employing the *k*-fold cross-validation approach. The accuracy of the classifier model measured by the validation method is 0.9695 on average.

Table 1 Overview of the datasets used in our experiments

	Malicious samples			Benign samples
	VirusShare [75]		AMD [79]	Google Play Store [24]
	2018	2019		
Training dataset	9000	N/A	N/A	9000
Test dataset	1000	653	23,039	1000

Table 2 High-weight API call graphs discovered by Grad-CAM

No.	Weight score	API call graph
1	3.51E−06	android.content.pm − > java.lang
2	3.48E−06	android.text.style − > java.lang
3	3.27E−06	java.security.cert − > java.lang
4	3.22E−06	android.graphics.drawable − > java.lang
5	3.19E−06	java.security − > java.lang
6	3.09E−06	android.webkit − > android.util
7	2.90E−06	android.accounts − > java.lang
8	2.90E−06	android.webkit − > android.widget
9	2.78E−06	org.xmlpull.v1 − > java.lang
...	...	...
4312	1.47E−18	com.google.firebase − > javax.xml.parsers

Table 3 Performance evaluation results of MAPAS and CNN

	CPU (%)	GPU (MiB)	RAM (MB)	Time (s)	Accuracy (%)
MAPAS	1.0925	None	157.543	21.1799	93.2
CNN	1.5425	10,590	2070.3692	15.9171	83

Finding high-weight Features with Grad-CAM After generating the classifier model, we used Grad-CAM [69] to observe high-weight API call graphs. The number of API call graphs that have a positive weight score is 4312 as shown in Table 2. Based on these 4312 API call graphs, MAPAS finds malicious applications.

To pick a threshold, we measured the Jaccard similarity between the high-weight API call graphs and API call graphs extracted from malicious applications and benign ones. As result, the similarity score is 0.561 and 0.2996, respectively. We used the average value (0.4303) of two scores as a threshold value for detecting malware. In this work, MAPAS can avoid biased results by using the average value. In other words, MAPAS avoid false negatives that when the classifier detections the application is benign when it is actually malware and false positives that is classifying the application is malware when it is actually benign by using average score.

5.3 Performance evaluation of MAPAS with the CNN classifier model

MAPAS uses the Jaccard similarity algorithm as a classifier to detect malware. We evaluated the performance and the usage of computing resources of MAPAS's malware detection process. Also, we measured the performance and the usage of computing resources of the classifier model generated by CNN. For this evaluation, we used 1000 malicious applications and 1000 benign applications of the test dataset as shown in Table 1.

Table 3 shows the experimental results. To classify 2000 applications, MAPAS took 21.18 s (1.059 ms on average) on a single core. The classifier model processed them in 15.92 s (0.796 ms on average) by using one GPU. It is worth noting that, when we used the classifier model without using a GPU, we could not finish processing 2000 applications within 24 h. In addition, as in Table 3, the classifier model used 10,590 MiB of GPU memory and about 2070 MB of RAM (1214.16% more than MAPAS). We, also, measured the detection accuracy. The CNN classifier model showed 11% lower detection rate than MAPAS.

5.4 Performance evaluation of MAPAS with MaMaDroid

We compare the performance of MAPAS to previous work (MaMaDroid [61]). Similar to MAPAS, MaMaDroid uses API call graphs of malicious applications to detect them. To compare the performance, MAPAS and MaMaDroid [61] created a classifier by using 9000 benign applications and 9000 malicious ones in the training dataset. MaMaDroid converted API call graphs into Markov chain [59] and created a classifier by learning 198,916 features. On the other hand, MAPAS used unique 21,659 API call graphs for creating a classifier. By default, MaMaDroid uses random forest (RF) [13] and k-nearest neighbors (k-NN) [17]. Also, in this evaluation, we did not use a GPU but only a CPU for both MaMaDroid and MAPAS.

Performance of the learning process Figure 7 shows the evaluation results of learning phases in each system. MaMaDroid +CNN used about 1214% of RAM more than MAPAS for the learning phase (MAPAS used 2.26 GB of RAM and MaMadroid+CNN used 34 GB of RAM). Also, MaMaDroid +CNN spent 5.45 times as much time as MAPAS did to

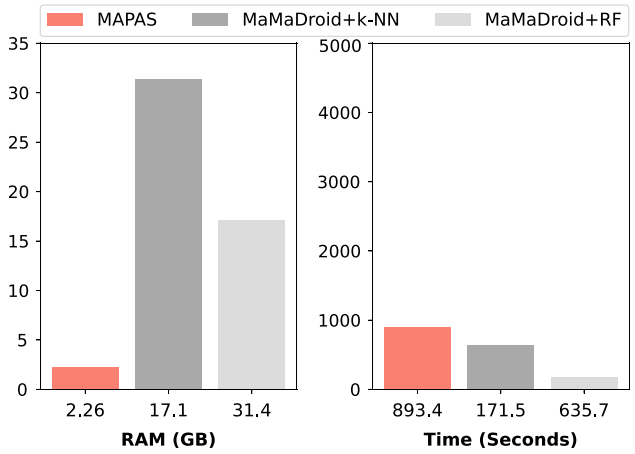

Fig. 7 Performance evaluation results of the learning process of MAPAS and MaMaDroid

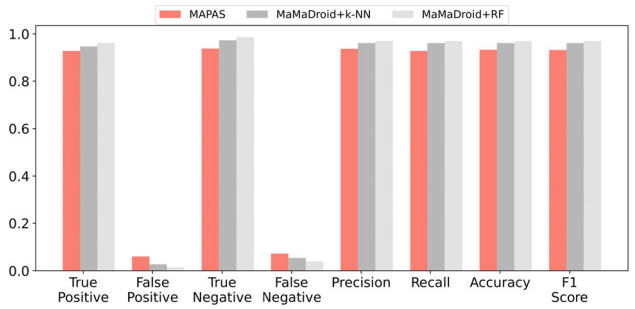

Fig. 8 Accuracy of classification results of MAPAS and MaMaDroid

finish learning the dataset. However, MaMaDroid+RF and MapaDroid+k-NN finished the learning phase faster than MAPAS, even though they used much more memory than MAPAS.

Performance of the classification process To evaluate the classification process of MAPAS and MaMaDroid, we used each system for classifying 2000 applications in the test dataset. The evaluation results are shown in Figs. 8 and 9. Overall, MaMaDroid using the random forest algorithm (MaMaDroid+RF) showed the best accuracy as in Fig. 8. MAPAS achieves about 3% lower accuracy than MaMaDroid+RF. However, MAPAS showed the best performance in terms of the execution time and the lowest RAM usage as illustrated in Fig. 9. To be specific, MAPAS can classify applications 76.4% and 145.8% faster than MaMaDroid+RF and MaMaDroid+k-NN, using much lower memory (MAPAS used memory around ten times lower than MaMaDroid+RF).

Detecting malware of various categories We evaluated the effectiveness of MAPAS and MaMaDroid+RF for detecting Android malware in 70 categories defined by Wei et al. [79]. The measurement results are shown in Table 4. MAPAS showed about 99% accuracy for 70 malware categories on average. This result demonstrates that MAPAS can generally detect any type of malware with high accuracy. On the other hand, MaMaDroid detected malware with 69% accuracy on average. Specifically, MaMaDroid showed high accuracy for detecting malware in categories such as BankBot, Univert, Utchi, FakeDoc, but it cannot accurately detect malware in categories such as Bankun, FakePlayer, FakeUpdates, Leech, Nandrobox, SlemBunk, Smskey and SmsZombie. Bankun, Fakeplayer are Trojan-type that hides in the normal flow and abruptly execute [79]. This means Trojan-type does not affect the current state to the next state transition. Thus, MaMaDroid cannot detect Trojan-type malware due to it uses Markov chain which relies on the current state and next state probability of transitions.

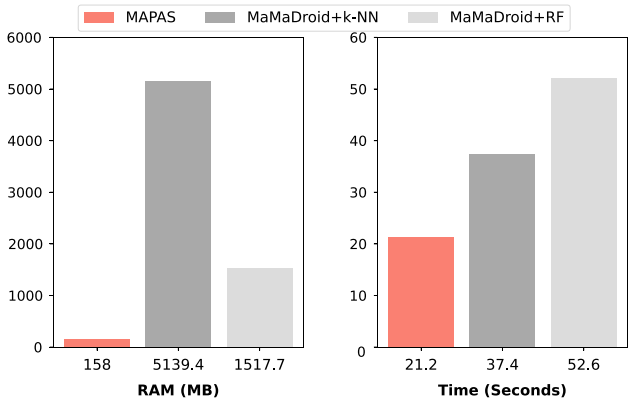

Fig. 9 Performance evaluation results of the classification process of MAPAS and MaMaDroid

Detecting unknown malware We evaluated the performance of detecting unknown malicious applications by using MAPAS and MaMaDroid+RF. To this end, we collected malware, released later than applications in the training dataset, from VirusShare [75]. As in Table 4, MAPAS showed 91% accuracy for detecting unkown malware, which is 6% higher than MaMaDroid.

6 Related work

DroidRisk [76] and Dini et al. [19] used permissions as features of malware for detecting them by using Analytic Hierarchy Process (AHP). Also, the following work employed permissions as features of malware but used different learning algorithms. Peng et al. [64] used Naive Bayes, Zarni et al. [10], Li et al. [46], FAMOUS [42] and Pehlivan et al. [63] used tree-based machine learning algorithms to detect malware. Ganesh et al. [23] used CNN.

Santos et al. [66], TinyDroid [16], McLaughlin el al. [54] and Deeprefiner [85] classified malware based on their opcode (bytecode instructions) using various machine learning algorithms such as SVM, k-NN, decision tree, naive

Table 4 Accuracy of MAPAS and MaMaDroid for detecting malware in 70 categories

Category	Accuracy (%)		Category	Accuracy (%)	
	MAPAS	MaMaDroid+RF		MAPAS	MaMaDroid+RF
Airpush	99.99	71	FakeUpdates	100	20
AndroRAT	100	84	Finspy	100	100
Andup	100	78	Fjcon	100	100
Aples	100	0	Fobus	100	100
BankBot	92.61	100	Fusob	100	100
Bankun	100	16	GingerMaster	100	73
Boqx	99.53	81	GoldDream	100	81
Boxer	100	0	Gorpo	100	100
Cova	64.71	65	Gumen	100	99
Dowgin	100	97	Jisut	99.79	13
DroidKungFu	99.82	87	Kemoge	100	93
Erop	100	100	Koler	100	81
FakeAngry	100	20	Ksapp	100	83
FakeAV	100	80	Kuguo	100	100
FakeDoc	100	100	Kyview	100	95
FakeInst	98.06	43	Leech	100	26
FakePlayer	100	24	Lnk	100	600
FakeTimer	100	0	Lotoor	94.51	60
Mecor	100	0	SpyBubble	100	50
Minimob	100	75	Stealer	100	100
Mmarketpay	100	93	Steek	100	100
MobileTX	100	0	Svpeng	100	100
Mseg	100	86	Tesbo	100	100
Mtk	100	100	Triada	100	95
Nandrobox	100	42	Univert	100	100
Ogel	100	100	UpdtKiller	100	62
Opfake	100	88	Utchi	100	100
Penetho	100	56	Vidro	100	87
Ramnit	100	12	VikingHorde	85.71	57
Roop	100	100	Vmvol	100	100
RuMMs	100	73	Winge	100	58
SimpleLocker	93.67	70	Youmi	100	93
SlemBunk	100	18	Zitmo	100	46
Smskey	100	43	Ztorg	100	90
SmsZombie	100	0	Average accuracy	98.98	68.63
Spambot	100	80			

Bayes, Bayesian networks, multilayer perceptron (MLP) and long short-term memory models (LSTM).

Droidapiminer [2] detected malware with machine learning algorithms such as k-NN, Iterative Dichotomiser 3 (ID3), SVM and C4.5 by using frequently used APIs in malware. Nix et al. [58] and MalDozer [34] also attempted to detect malware by using CNN, LSTM, SVM and Naive Bayes based on APIs used in malware. Droiddelver [30] detected malware by analyzing API call blocks of them with deep belief network (DBN) and restricted Boltzmann machine (RBN) algorithms.

On the other hand, Yerima et al. [86], Droidmat [82], Drebin [7], DroidDolphin [84], Chan et al. [15] used complex features (i.e., using more than two different types of features such as Permission, API, Opcode) with various machine learning algorithms. DroidDeepLearner [77], Hou et al. [31], Li et al. [49], Li et al. [47], Zhang et al. [88], kim et al. [38] proposed deep learning-based malware detection systems based on complex features. The above studies showed a

high detection rate by detecting malware with features containing various information but require a lot of computing resources.

The closet related work to this paper is MaMaDroid [61] that used Markov chain [59] to calculate the probability of transition from the current state (Sources) to another state (Sinks) from API call graphs used in malicious applications. MaMaDroid, then, utilized k-NN and random forest algorithms to train the Markov chains and to generate a classifier model. Besides, DeepFlow [92] and EveDroid [45] also used API call graphs for detecting malware. They especially focused on detecting newly emerging malicious applications by using a deep learning algorithm.

7 Conclusion

In this paper, we proposed MAPAS, an effective and efficient malware detection approach. MAPAS analyzes common features of API call graphs extracted from malicious applications by using a deep learning algorithm. Then, it detects malware based on the features with a lightweight classifier for the efficiency. Our evaluation results showed that MAPAS outperforms a state-of-the-art approach, MaMaDroid [61], in terms of the usage of computing resources and the accuracy for detecting unknown malware. Also, MAPAS can generally detect any type of malware with high accuracy.

Funding This work was supported in part by Institute for Information & communication Technology Planning & evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2017-0-00168, Automatic Deep Malware Analysis Technology for Cyber Threat Intelligence). Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Republic of Korea Government or any agency thereof.

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.

Appendix A: CNN

Convolutional neural network (CNN) is an algorithm of artificial neural networks using convolution arithmetic. The convolution arithmetic is operations by extracting the regional information of the data in the filter (or kernel) is moving. The filter calculates the convolution while moving the input data at a set interval. At this time, the interval of the filter moves is called the stride. The output data of result from convolution arithmetic is called a feature map. Feature map uses ReLU among activation functions to extract only positive values. After that, a new layer is created by the pooling. Pooling is a method that reducing the size of a feature map and emphasizing feature information. As the number of filters increases in the convolution arithmetic, the number of feature maps increases. Therefore, as feature maps increase, there is a risk of many memory use and overfitting due to many features. As we already expressed, to prevent convolution arithmetic problems, CNN uses pooling. There are max pooling and average pooling in pooling. Max pooling reduces the size by leaving only the largest value among the feature map values. Average pooling reduces the size by calculating the average from the feature map values. After that, the extracted output data transform one dimension. The output data of one dimension is called a fully connected layer (FC). In FC, the result value is classified using the softmax among activation functions.

CNN was originally designed for processing images. However, recently CNN is usually used for the natural language processing [35].

Appendix B: CAM

Because a deep learning algorithm operates in a black-box manner, it is important to interpret values affected the classification results in a deep learning model. Therefore, in recent years, many interpretation approaches were proposed to identify features that have an important influence on classification results in the deep learning model [4,29]. Class activation map (CAM) among interpretation approaches is used to the CNN algorithm [91]. CAM uses global average pooling (GAP) instead of FC to extract the feature information value of CNN. The formula of CAM is as follows.

$$L^c_{CAM}(x, y) = \sum_k w^c_k f_k(x, y) \tag{B.1}$$

However, CAM should be used by replacing FC with GAP.

Grad-CAM Gradient-weighted class activation map (Grad-CAM) [69] does not use GAP and extract features that affect the result. Grad-CAM uses the gradient value about the class of the last convolution layer by backpropagation to calculate the value of CAM. The formula of Grad-CAM is as follows.

$$L^c_{Grad\,CAM} = ReLU\left(\sum_k \alpha^c_k f_k(x, y)\right) \tag{B.2}$$

References

1. 2020 McAfee Mobile Threat Report. McAfee Labs (2020)
2. Aafer, Y., Du, W., Yin, H.: Droidapiminer: Mining api-level features for robust malware detection in android. In: International Conference on Security and Privacy in Communication Systems, pp. 86–103. Springer, Berlin (2013)
3. Accard, P.: The distribution of the flora in the alpine zone. New Phytol. **11**(2), 37–50 (1912)
4. Adadi, A., Berrada, M.: Peeking inside the black-box: a survey on explainable artificial intelligence (XAI). IEEE Access **6**, 52138–52160 (2018)
5. Afonso, V.M., de Amorim, M.F., Grégio, A.R.A., Junquera, G.B., de Geus, P.L.: Identifying android malware using dynamically obtained features. J. Comput. Virol. Hacking Tech. **11**(1), 9–17 (2015)
6. Allix, K., Bissyandé, T.F., Klein, J., Traon, Y.L.: Androzoo: collecting millions of android apps for the research community. In: 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR), pp. 468–471. IEEE (2016)
7. Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K., CERT, S.: Drebin: effective and explainable detection of android malware in your pocket. In: Ndss, vol. 14, pp. 23–26 (2014)
8. Arzt, S.: Static data flow analysis for android applications (2017)
9. Arzt, S., Rasthofer, S., Fritz, C., Bodden, E., Bartel, A., Klein, J., Traon, Y.L., Octeau, D., McDaniel, P.: Flowdroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. Acm Sigplan Notices **49**(6), 259–269 (2014)
10. Aung, Z., Zaw, W.: Permission-based android malware detection. Int. J. Sci. Technol. Res. **2**(3), 228–234 (2013)
11. Avdiienko, V., Kuznetsov, K., Gorla, A., Zeller, A., Arzt, S., Rasthofer, S., Bodden, E.: Mining apps for abnormal usage of sensitive data. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, vol. 1, pp. 426–436. IEEE (2015)
12. Bläsing, T., Batyuk, L., Schmidt, A.-D., Camtepe, S.A., Albayrak, S.: An android application sandbox system for suspicious software detection. In: 2010 5th International Conference on Malicious and Unwanted Software, pp. 55–62. IEEE (2010)
13. Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001)
14. Burguera, I., Zurutuza, U., Nadjm-Tehrani, S.: Crowdroid: behavior-based malware detection system for android. In: Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices, pp. 15–26 (2011)
15. Chan, P.P., Song, W.-K.: Static detection of android malware by using permissions and API calls. In: 2014 International Conference on Machine Learning and Cybernetics, vol. 1, pp. 82–87. IEEE (2014)
16. Chen, T., Mao, Q., Yang, Y., Lv, M., Zhu, J.: Tinydroid: a lightweight and efficient model for android malware detection and classification. In: Mobile Information Systems, 2018 (2018)
17. Cover, T., Hart, P.: Nearest neighbor pattern classification. IEEE Trans. Inf. Theory **13**(1), 21–27 (1967)
18. Desnos, A., Gueguen, G.: Android: from reversing to decompilation. In: Proc. of Black Hat Abu Dhabi, pp. 77–101 (2011)
19. Dini, G., Martinelli, F., Matteucci, I., Petrocchi, M., Saracino, A., Sgandurra, D.: Risk analysis of android applications: a user-centric solution. Futur. Gener. Comput. Syst. **80**, 505–518 (2018)
20. Fan, M., Liu, J., Luo, X., Chen, K., Chen, T., Tian, Z., Zhang, X., Zheng, Q., Liu, T.: Frequent subgraph based familial classification of android malware. In: 2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE), pp. 24–35. IEEE (2016)
21. Feng, Y., Anand, S., Dillig, I., Aiken, A.: Apposcopy: semantics-based detection of android malware through static analysis. In: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 576–587 (2014)
22. Ferrante, A., Malek, M., Martinelli, F., Mercaldo, F., Milosevic, J.: Extinguishing ransomware-a hybrid approach to android ransomware detection. In: International Symposium on Foundations and Practice of Security, pp. 242–258. Springer, Berlin (2017)
23. Ganesh, M., Pednekar, P., Prabhuswamy, P., Nair, D.S., Park, Y., Jeon, H.: CNN-based android malware detection. In: 2017 International Conference on Software Security and Assurance (ICSSA), pp. 60–65. IEEE (2017)
24. Google Play Store. https://play.google.com/store/apps. Accessed November (2019)
25. Gordon, M.I., Kim, D., Perkins, J.H., Gilham, L., Nguyen, N., Rinard, M.C.: Information flow analysis of android applications in droidsafe. In: NDSS, vol. 15, p. 110 (2015)
26. Ham, Y.J., Lee, H.-W.: Detection of malicious android mobile applications based on aggregated system call events. Int. J. Comput. Commun. Eng. **3**(2), 149 (2014)
27. Ham, Y.J., Moon, D., Lee, H.-W., Lim, J.D., Kim, J.N.: Android mobile application system call event pattern analysis for determination of malicious attack. Int. J. Secur. Appl. **8**(1), 231–246 (2014)
28. HCL AppScan. https://www.hcltechsw.com/appscan/. Accessed March (2021)
29. Hossain, M.S., Amin, S.U., Alsulaiman, M., Muhammad, G.: Applying deep learning for epilepsy seizure detection and brain mapping visualization. ACM Trans. Multimed. Comput. Commun. Appl. (TOMM) **15**(1s), 1–17 (2019)
30. Hou, S., Saas, A., Ye, Y., Chen, L.: Droiddelver: an android malware detection system using deep belief network based on api call blocks. In: International Conference on Web-Age Information Management, pp. 54–66. Springer, Berlin (2016)
31. Hou, S., Saas, A., Chen, L., Ye, Y., Bourlai, T.: Deep neural networks for automatic android malware detection. In: Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017, pp. 803–810 (2017)
32. Isohara, T., Takemori, K., Kubota, A.: Kernel-based behavior analysis for android malware detection. In: 2011 Seventh International Conference on Computational Intelligence and Security, pp. 1011–1015. IEEE (2011)
33. Jing, Y., Ahn, G.-J., Zhao, Z., Hu, H.: Riskmon: continuous and automated risk assessment of mobile applications. In: Proceedings of the 4th ACM Conference on Data and Application Security and Privacy, pp. 99–110 (2014)
34. 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)
35. Kim, Y.: Convolutional neural networks for sentence classification. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25–29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL, pp. 1746–1751. ACL (2014)

36. Kim, G., Lee, S., Kim, S.: A novel hybrid intrusion detection method integrating anomaly detection with misuse detection. Expert Syst. Appl. **41**(4), 1690–1700 (2014)
37. Kim, H., Cho, T., Ahn, G.-J., Yi, J.H.: Risk assessment of mobile applications based on machine learned malware dataset. Multimed. Tools Appl. **77**(4), 5027–5042 (2018)
38. Kim, T., Kang, B., Rho, M., Sezer, S., Im, E.G.: A multimodal deep learning method for android malware detection using various features. IEEE Trans. Inf. Forensics Secur. **14**(3), 773–788 (2018)
39. Kim, K., Ko, E., Kim, J., Yi, J.H.: Intelligent malware detection based on hybrid learning of API and ACG on android. J. Internet Ser. Inf. Secur. **9**(4), 39–48 (2019)
40. Kim, K., Kim, J., Ko, E., Yi, J.H.: Risk assessment scheme for mobile applications based on tree boosting. IEEE Access **8**, 48503–48514 (2020)
41. Kong, D., Cen, L., Jin, H.: Autoreb: automatically understanding the review-to-behavior fidelity in android applications. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, pp. 530–541 (2015)
42. Kumar, A., Kuppusamy, K., Aghila, G.: Famous: Forensic analysis of mobile devices using scoring of application permissions. Futur. Gener. Comput. Syst. **83**, 158–172 (2018)
43. Le, Q., Mikolov, T.: Distributed representations of sentences and documents. In: International Conference on Machine Learning, pp. 1188–1196. PMLR (2014)
44. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)
45. Lei, T., Qin, Z., Wang, Z., Li, Q., Ye, D.: Evedroid: event-aware android malware detection against model degrading for iot devices. IEEE Internet Things J. **6**(4), 6668–6680 (2019)
46. Li, Y., Li, Y., Yan, H., Liu, J.: Deep joint discriminative learning for vehicle re-identification and retrieval. In: 2017 IEEE International Conference on Image Processing (ICIP), pp. 395–399. IEEE (2017)
47. 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. IEEE (2018)
48. Li, J., Sun, L., Yan, Q., Li, Z., Srisa-An, W., Ye, H.: Significant permission identification for machine-learning-based android malware detection. IEEE Trans. Ind. Inf. **14**(7), 3216–3225 (2018)
49. Li, W., Wang, Z., Cai, J., Cheng, S.: An android malware detection approach using weight-adjusted deep learning. In: 2018 International Conference on Computing, Networking and Communications (ICNC), pp. 437–441. IEEE (2018)
50. Liu, P., Wang, W., Luo, X., Wang, H., Liu, C.: Nsdroid: efficient multi-classification of android malware using neighborhood signature in local function call graphs. Int. J. Inf. Secur. 1–13 (2020)
51. Ma, Z., Ge, H., Liu, Y., Zhao, M., Ma, J.: A combination method for android malware detection based on control flow graphs and machine learning algorithms. IEEE Access **7**, 21235–21245 (2019)
52. Maiorca, D., Ariu, D., Corona, I., Aresu, M., Giacinto, G.: Stealth attacks: an extended insight into the obfuscation effects on android malware. Comput. Secur. **51**, 16–31 (2015)
53. Martín, A., Calleja, A., Menéndez, H.D., Tapiador, J., Camacho, D.: Adroit: android malware detection using meta-information. In: 2016 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 1–8. IEEE (2016)
54. McLaughlin, N., del Rincon, J.M., Kang, B., Yerima, S., Miller, P., Sezer, S., Safaei, Y., Trickel, E., Zhao, Z., Doupé, A., Ahn, G.J.: Deep android malware detection. In: Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, pp. 301–308 (2017)
55. Mikolov, T., Sutskever, I., Chen, K., Corrado, G., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Proceedings of the 26th International Conference on Neural Information Processing Systems, vol. 2, pp. 3111–3119 (2013)
56. Mohr, M., Graf, J., Hecker, M.: Jodroid: Adding android support to a static information flow control tool. In: Software Engineering (Workshops), pp. 140–145. Citeseer (2015)
57. Molnar, C.: Interpretable machine learning. Lulu. com (2020)
58. Nix, R., Zhang, J.: Classification of android apps and malware using deep neural networks. In: 2017 International Joint Conference on Neural Networks (IJCNN), pp. 1871–1878. IEEE (2017)
59. Norris, J.R., Norris, J.R., Norris, J.R.: Markov Chains, vol. 2. Cambridge University Press, Cambridge (1998)
60. Octeau, D., McDaniel, P., Jha, S., Bartel, A., Bodden, E., Klein, J., Traon, Y.L.: Effective inter-component communication mapping in android: an essential step towards holistic security analysis. In: 22nd USENIX Security Symposium (USENIX Security 13), pp. 543–558 (2013)
61. 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. (TOPS) **22**(2), 1–34 (2019)
62. Pandita, R., Xiao, X., ang, W., Enck, W., Xie, T.: WHYPER: Towards automating risk assessment of mobile applications. In: 22nd USENIX Security Symposium (USENIX Security 13), pp. 527–542 (2013)
63. Pehlivan, U., Baltaci, N., Acartürk, C., Baykal, N.: The analysis of feature selection methods and classification algorithms in permission based android malware detection. In: 2014 IEEE Symposium on Computational Intelligence in Cyber Security (CICS), pp. 1–8. IEEE (2014)
64. Peng, W., Huang, L., Jia, J., Ingram, E.: Enhancing the Naive Bayes spam filter through intelligent text modification detection. In: 2018 17th IEEE International Conference on Trust, Security and Privacy in Computing And Communications/12th IEEE International Conference on Big Data Science and Engineering (TrustCom/BigDataSE), pp. 849–854. IEEE (2018)
65. Qiu, L., Wang, Y., Rubin, J.: Analyzing the analyzers: Flowdroid/iccta, amandroid, and droidsafe. In: Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 176–186 (2018)
66. Santos, I., Brezo, F., Ugarte-Pedrero, X., Bringas, P.G.: Opcode sequences as representation of executables for data-mining-based unknown malware detection. Inf. Sci. **231**, 64–82 (2013)
67. Saracino, A., Sgandurra, D., Dini, G., Martinelli, F.: Madam: effective and efficient behavior-based android malware detection and prevention. IEEE Trans. Dependable Secur. Comput. **15**(1), 83–97 (2016)
68. Schmidhuber, J.: Deep learning in neural networks: an overview. Neural Netw. **61**, 85–117 (2015)
69. Selvaraju, R.R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., Batra, D.: Grad-cam: visual explanations from deep networks via gradient-based localization. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 618–626 (2017)
70. Shijo, P., Salim, A.: Integrated static and dynamic analysis for malware detection. Proc. Comput. Sci. **46**, 804–811 (2015)
71. Strubell, E., Ganesh, A., McCallum, A.: Energy and policy considerations for deep learning in nlp. In: Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pp. 3645–3650 (2019)
72. Suarez-Tangil, G., Tapiador, J.E., Peris-Lopez, P., Blasco, J.: Dendroid: a text mining approach to analyzing and classifying code structures in android malware families. Expert Syst. Appl. **41**(4), 1104–1117 (2014)
73. Talha, K.A., Alper, D.I., Aydin, C.: APK auditor: permission-based android malware detection system. Digit. Investig. **13**, 1–14 (2015)
74. Tong, F., Yan, Z.: A hybrid approach of mobile malware detection in android. J. Parallel Distrib. Comput. **103**, 22–31 (2017)

75. VirusShare. https://virusshare.com/. Accessed November (2019)
76. Wang, Y., Zheng, J., Sun, C., Mukkamala, S.: Quantitative security risk assessment of android permissions and applications. In: IFIP Annual Conference on Data and Applications Security and Privacy, pp. 226–241. Springer, Berlin (2013)
77. Wang, Z., Cai, J., Cheng, S., Li, W.: Droiddeeplearner: identifying android malware using deep learning. In: 2016 IEEE 37th Sarnoff Symposium, pp. 160–165. IEEE (2016)
78. Wang, W., Zhao, M., Wang, J.: Effective android malware detection with a hybrid model based on deep autoencoder and convolutional neural network. J. Ambient. Intell. Humaniz. Comput. **10**(8), 3035–3043 (2019)
79. Wei, F., Li, Y., Roy, S., Zhou, X.O.W.: Deep ground truth analysis of current android malware. In: International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, pp. 252–276. Springer, Berlin (2017)
80. Wei, F., Roy, S., Ou, X.: Amandroid: a precise and general inter-component data flow analysis framework for security vetting of android apps. ACM Trans. Priv. Secur. (TOPS) **21**(3), 1–32 (2018)
81. Wook Jang, J., Kang, H., Woo, J., Mohaisen, A., Kim, H.K.: Andro-dumpsys: anti-malware system based on the similarity of malware creator and malware centric information. Comput. Secur. **58**, 125–138 (2016)
82. Wu, D.-J., Mao, C.-H., Wei, T.-E., Lee, H.-M., Wu, K.-P.: Droidmat: android malware detection through manifest and API calls tracing. In: 2012 Seventh Asia Joint Conference on Information Security, pp. 62–69. IEEE (2012)
83. Wu, S., Wang, P., Li, X., Zhang, Y.: Effective detection of android malware based on the usage of data flow APIs and machine learning. Inf. Softw. Technol. **75**, 17–25 (2016)
84. Wu, W.-C., Hung, S.-H.: Droiddolphin: a dynamic android malware detection framework using big data and machine learning. In: Proceedings of the 2014 Conference on Research in Adaptive and Convergent Systems, pp. 247–252 (2014)
85. Xu, K., Li, Y., Deng, R.H., Chen, K.: Deeprefiner: multi-layer android malware detection system applying deep neural networks. In: 2018 IEEE European Symposium on Security and Privacy (EuroS&P), pp. 473–487. IEEE (2018)
86. Yerima, S.Y., Sezer, S., Muttik, I.: Android malware detection using parallel machine learning classifiers. In: 2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies, pp. 37–42. IEEE (2014)
87. Yuan, Z., Lu, Y., Xue, Y.: Droiddetector: android malware characterization and detection using deep learning. Tsinghua Sci. Technol. **21**(1), 114-123 (2016)
88. Zhang, Y., Yang, Y., Wang, X.: A novel android malware detection approach based on convolutional neural network. In: Proceedings of the 2nd International Conference on Cryptography, Security and Privacy, pp. 144–149 (2018)
89. Zhang, H., Luo, S., Zhang, Y., Pan, L.: An efficient android malware detection system based on method-level behavioral semantic analysis. IEEE Access **7**, 69246–69256 (2019)
90. Zhou, Y., Wang, Z., Zhou, W., Jiang, X.: Hey, you, get off of my market: detecting malicious apps in official and alternative android markets. In NDSS **25**, 50–52 (2012)
91. Zhou, B., Khosla, A., Lapedriza, A., Oliva, A., Torralba, A.: Learning deep features for discriminative localization. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2921–2929 (2016)
92. Zhu, D., Jin, H., ang, Y., Wu, D., Chen, W.: Deepflow: deep learning-based malware detection by mining android application for abnormal usage of sensitive data. In: 2017 IEEE Symposium on computers and Communications (ISCC), pp. 438–443. IEEE (2017)