Malware Analysis by Combining Multiple Detectors and Observation Windows

Massimo Ficco

Abstract—Malware developers continually attempt to modify the execution pattern of malicious code hiding it inside apparent normal applications, which makes its detection and classification challenging. This article proposes an ensemble detector, which exploits the capabilities of the main analysis algorithms proposed in the literature designed to offer greater resilience to specific evasion techniques. In particular, the article presents different methods to optimally combine both generic and specialized detectors during the analysis process, which can be used to increase the unpredictability of the detection strategy, as well as improve the detection rate in presence of unknown malware families and provide better detection performance in the absence of a constant re-training of detector needed to cope with the evolution of malware. The paper also presents an *alpha-count* mechanism that explores how the length of the observation time window can affect the detection accuracy and speed of different combinations of detectors during the malware analysis. An extended experimental campaign has been conducted on both an open-source sandbox and an Android smartphone with different malware datasets. A trade-off among performance, training time, and mean-time-to-detect is presented. Finally, a comparison with other ensemble detectors is also presented.

Index Terms—Malware detection, malware evasion, feature extraction, diversity of detection algorithms, API sequence, memory dump, correlation, deep learning, ensemble detector, malware activation, training time, observation windows, mean-time-to-detect

1 Introduction

According to the Kaspersky report, the number of new malware produced in 2017 reached 360,000 and grows by 11 percent every year [1]. Recently, experts at AV-TEST Institute had counted around 1.8 million new malicious applications in the first half of 2020 [2]. Malware developers continually produce new malware variants and use several evasive techniques to hide the malicious behaviour in legitimate applications and delay the detection process. Each new variant of malware could evade signature-based detection methods used by most of today's Anti-Virus software, until a new malware signature is produced [3].

In the last year, several dynamic analysis detectors have been presented in the literature. Most of the proposed detection techniques exploit multiple application features during the analysis process (mainly based on the frequency or presence of such features during the malware execution). Combining multiple features, it is possible to represent malware more comprehensively and provide better detection performance, especially if each detector is specialized for detecting a different malware family [4]. Moreover, although arbitrarily complex but deterministic classifiers can always be reverse-engineered, or however evaded, recent results in adversarial evasion show that the use of multiple diverse detectors and stochastic mechanisms for unpredictably switching between them makes the ensemble detector more resilient to both reverse-engineering and evasion. The resilience of the detection technique increases with the number of diversities introduced [5].

Therefore, starting from these assumptions, this paper proposes an ensemble detector that exploits diversity in the detection algorithms, rather than just differentiating the vector features, and in particular, it presents an extensive experimental campaign that shows the detection performance as a function of the incorporated detectors and the length of the observation time window, useful to implement multiple schemes to optimally combine detectors and increase the unpredictability of the detection strategy. Moreover, it discusses how the effects of a larger observation window and the switching between different detector combinations during the analysis process can improve the detection accuracy and speed. In particular, the main contributes are:

1) An analysis of the main dynamic detectors used in the literature, which exploit hardware or software features (including memory dump, API calls, and network activity). They can be exploited to describe malware from multiple dimensions. Moreover, for each of them, the most significant detection algorithm has been considered as a reference, according to its capability to offer greater resilience to specific evasion techniques (such as anti-emulation and anti-debugging, insertion and replacement of API calls in the malicious code, etc.).
2) Assuming that a switching between different sets of specialized and generic detectors during the analysis process can improve the unpredictability of the detection technique, this paper presents methods to optimally combine detectors, as well as shows the

• *The author is with the Department of Engineering, University of Campania Luigi Vanvitelli, 81031 Aversa, CE, Italy. E-mail: massimo.ficco@unicampania.it.*

Manuscript received 1 Oct. 2020; revised 3 May 2021; accepted 9 May 2021. Date of publication 19 May 2021; date of current version 10 May 2022. Recommended for acceptance by M. Tehranipoor. Digital Object Identifier no. 10.1109/TC.2021.3082002

achieved detection performance as a function of the number of incorporated specialized and generic detectors, the kind of detection algorithm, and the observation window. Moreover, a trade-off among performance, training time, and mean-time-to-detect is also presented.

3) An *alpha-count* mechanism is proposed and used to explore how the effects of a larger observation time window and multiple computation periods can affect the detection accuracy and speed of different combinations of detectors during the analysis process.
4) Finally, a comparative analysis with other ensemble detectors proposed in the literature is shown.

The adopted training dataset includes malicious apps belonging to several families of malware collected over different shared datasets. The testing set is composed of malware families also not included in the training dataset, with a mix of older and newer apps. It is used to evaluate the effectiveness of different ensemble detectors in detecting unknown malware families, as well as verify its effectiveness with respect to the evolution of malware over time. The experiments have been conducted on both an open-source sandbox and a real smartphone. The results have shown that a combination of generic and specialized detectors can be more resilient to the malware evolution with respect to use only specialized detectors, as well as more performant with respect to the presence in the testing set of unknown malware families not belonging to the same types that the specialized detectors have been designed for. Moreover, the experiments show that the use of a larger observation time window and the switching between different combinations of detectors during the analysis process can increase the detection accuracy and speed, as well as offer further schemes for diversifying the ensemble detector during the analysis process, useful to increase the unpredictability of the detection strategy.

The rest of the paper is organized as follows: Section 2 presents the related work. Section 3 explains the overall ensemble learning-based approach. Section 4 presents the features and the algorithms used for malware classification. Experimental results are shown in Section 5. The discussion concerning the length of the observation windows is presented in Section 6. Conclusions and future work are summarized in Section 7.

2 Related Work

To discriminate benign software from malware, several static and dynamic features have been adopted to represent the application behaviour, including invoked API calls, opcode, performance counters, specific strings included in the source files, greyscale images of the executable files, network traffic, etc. Regarding dynamic analysis, classification based on API invoked by malicious code represents a lightweight approach for malware detection [6]. For example, Aafer *et al.* [7] generated the set of APIs used by a dataset of benign and malicious apps, and performed a frequency analysis to list out the API calls that were more frequent in the malware than in the benign set. On the other hand, relying on the most common and critical API calls involves constant retraining, in fact, this technique can be easily evaded by malware developers inserting dummy and redundant API calls in code. Techniques of removing, replacing, and adding redundant calls (or call invocations in the loops) in a random position in the code could be regarded as adding noises in the API call sequence of malware designed to evade the detection algorithm.

Therefore, several works capture the behavioural model of the app rather than their API call frequency [9]. Ki *et al.* [10] showed that certain API call patterns (malicious operations) are commonly included in malware even in different families, and they can be also used to detect new unknown malware. Moreover, Kim *et al.* [11] showed that adding dummy code or randomly reproducing existing code is ineffective against detectors based on API call sequence patterns, even at high levels of API insertions (e.g., 200 percent). Therefore, Cho *et al.* [12] proposed a malware similarity method based on API call sequences to detect variants of malware among the same group of malware. Elhadi *et al.* [13] presented a malware detector that exploited the API call graph. However, such matching algorithms have an NP-Complete problem and the detection phase is very slow due to their computational complexity. Therefore recently, several works used multiple sequence alignment algorithms (MSA). MSA matches API call sequences that have slightly different due to noises, by aligning the positions of API calls in sequences and calculating the similarity of a malicious code to the chains of behavioural sequences collected [12]. For example, Kim *et al.* [14] used MSA to implement malware detection and classification method collecting common patterns of the malware groups/families. On the other hand, malware developers may attempt to add, delete or replace API calls in a specific position of the malicious sequence (that do not affect the functionality of the malware), in order to produce new sequences that could reduce the effectiveness of the MSA based algorithm [11]. Alternative solutions adopted the Markov chain model, in which the feature vector of each app is represented by the probabilities of transitioning from one API call to another in the chain. This approach can be more resilient against evasion techniques [15]. Adding and removing/replacing API calls at specific points in the sequence would not significantly change the transition probabilities between two calls [11]. Moreover, it is very difficult to create sequences resulting in Markov chains similar to benign apps, which, at the same time, actually performing the malicious behaviour. Thus, MAMADROID [8] built a Markov model to represent the malware behaviour, by abstracting API calls to either the package name of the call or its family. VAED and CANDYMAN are tools that classify Android malware families by combining dynamic analysis and Markov chains [16], [17].

Alternatively, several research works have recently focused on hardware features, for example, exploiting memory dump and performance counters (e.g., memory behaviour, cache behavior, IPC, etc.) [18]. It can avoid the deficiencies of software-based techniques to face the malware evasion problem in the dynamic analysis [19]. In particular, Dai *et al.* [18] proposed a method based on memory dump data, which converted malware memory data dump binary files into greyscale images, and used multilayer perceptron to classify them. However, such a method cannot fully represent the behaviour of malware, and in particular, the classification accuracy could be insufficient.

Finally, in order to limit the effects of evasion techniques based on anti-debugger and anti-sandbox, network-based detection approaches have been proposed. They assume that malicious traffic is significantly more frequent or widespreading than normal traffic, resulting in a large number of packets or flows containing similar contents [20]. Chen *et al.* [21] designed an Android malware traffic behaviour monitoring scheme to capture traffic data generated by malware samples in a real Internet environment. They captured the DNS and HTTP network traffic in the first 5 minutes, and analysed the major compositions of the traffic data. Tenenboim *et al.* [22] showed the difference between malicious and benign applications in terms of network features. However, in these works, the number of samples analysed and the achieved classification accuracy is limited.

Each of the previous techniques exploited a single dynamic feature of malware. On the other hand, several works combined multiple features in order to reflect various behavioural aspects of applications. According to the authors, [23] was the first work that proposed a multimodal deep learning method for Android malware detection. However, it only used static features, including permission feature, opcode feature, API feature, shared library, component feature, and environmental features. [24], [25], [26] proposed different classifiers, which exploited multiple static and/or dynamic features. However, they focused on the detection accuracy and performance of the presented classifiers, which were implemented as generic detectors. Moreover, they only diversified the used features rather than the detection algorithms as well. In the same way, in [27], dynamic features were exploited to implement generic detectors combined using majority voting. However, none of the previous works specialized the implemented detectors for the different malware families. As shown in [4], each malware family can be characterized by different detection features. Smutz *et al.* [28] explored the use of diversified detectors to classify only a specific type of malware (i.e., evasive PDF malware) by machine learning algorithms. Khasawneh *et al.* [4] proved how the use of an ensemble detector composed only of specialized detectors can improve the detection accuracy of a specific type of malware with respect to generic detectors. In this paper is shown how the use of different detection algorithms (rather than just the existence of specific features or their frequency occurrence [4]) and the combined contribution of generic and specialized detectors can improve the detection of unknown malware families, as well as reduce the frequency of retraining with respect to the evolution of malware over time. Moreover, the paper evaluates how the use of a larger observation time window and the switching between different combinations of detectors during the analysis process can improve the detection performance and speed.

Finally, although each deterministic classifier (even if complex) can always be reverse-engineered and evaded [29], Khasawneh *et al.* [5] have proved how the use of multiple diverse classifiers and unpredictable switching strategies among them can make the detection technique more resilient to evasion attacks. Moreover, the resilience of detection techniques increases with the number and diversity of detectors. Therefore, this paper presents an extensive experimental campaign that shows the ensemble detection performance as a function of the number of the incorporated specialized and generic detectors, the kinds of detection algorithms adopted, and the length of the observation window, useful to optimally combine detectors and increase the unpredictability of the detection strategy.

3 Ensemble Learning Approach

The ensemble learning technique adopted in the literature for malware analysis consists in combining multiple independent detectors that are each trained to detect any type of malware [23]. On the other hand, as shown in [4], specialized detectors trained for specific malware types could perform better compared to a generic detector in classifying malware. However, our experimental results (Section 5) show that the ensemble detector composed only by specialized detectors does not always produce greater sensitivity than generic detectors, especially in presence of malware evolution. Therefore, a hybrid approach is proposed in this work, which uses both generic and specialized detectors. In particular, it will be presented the methods and the experimental results useful to optimally incorporate detectors, as well as a trade-off analysis among performance, training time, and detection time. Specifically, first, the most significant detection algorithms have been considered as a reference, according to their capability to offer greater resilience to specific evasion techniques. Then, for each algorithm, a specialized detector is separately trained with each single malware type available in the dataset (Backdoor, Worm, and Trojan), whereas the generic detectors are trained with all the dataset independently from malware families. Then, for each type of malware, the best specialized detectors that offer greater performance than the generic detectors are selected. Finally, in order to limit the number of specialized detectors, as well as improve the detection of malware types that are not accounted by the specialized detectors, we show a performance analysis of several ensemble detectors that use a different combination of generic and specialized detectors. All the detectors run in parallel in order to speed up the computation.

The stacking approach has been considered as a decision fusion function and used to combine the specialized and generic detectors. Collected malware features are separately processed in specialized and generic first-level neural networks, called 'low detectors'. Then, as shown in Fig. 1, the first-level detectors are connected to a second-level neural network, called 'meta-learner detector', for producing a final classification.

Each low detector is first trained. Then, the meta-learner detector is separately trained with the output of the previously learned initial networks. Finally, a validation dataset different by one used for training the low detectors is used for a meta-learner test. This learning strategy can enhance the overall accuracy of the model, reduce the training frequency needed to face malware evolution, as well as support stochastic switching between different low-detectors needed to increase the unpredictability of the detection strategy. Therefore, the following steps are followed:

- All malware and benign apps are executed in a controlled environment, and the software and hardware raw data that allows building the application behaviour is extracted (e.g., disk reading, API calls, virtual address space of the process, network data).

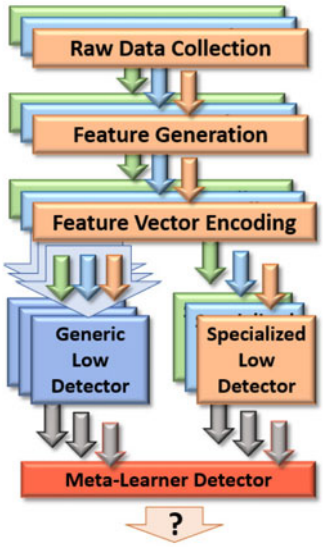

Fig. 1. The overall detection approach.

- The following features are derived from the processed apps: (i) the API call frequency, (ii) the API calls pattern, (iii) the probabilities of transitioning between two API calls, (iv) the memory data dump, and (v) the network messages, which can be modelled by call graphs, API sequences alignment, Markov chains, grayscale images, and traffic characterizations, respectively (Section 4). The collected features are converted into feature vectors to apply to the neural networks.
- A multimodal deep neural network is trained to fit the produced features. Specifically, in order to train the specialized detectors, the samples are categorized in malware types in advance, and then, separately processed in different neural networks (i.e., specialized low detectors). Moreover, the samples (regardless of the family they belong to) are all processed in the generic low detectors. The produced first-level classification is then processed in a neural network of second-level. Finally, several combinations of specialized and generic low detectors are tested to obtain the best classification accuracy (Section 5).

The multimodal neural network uses the produced five feature vectors, which are separately inputted to the low detectors. It consists of a combination of generic and specialized Deep Neural Networks (DNNs). The rectified linear units (ReLU) function is used as an activation function to prevent the vanishing gradient problem in training. Each low DNN consists of an initial layer and two hidden layers. Each layer is fully-connected to the previous layer, whose output passed to the next layer is expressed as

$$y^l = \phi(W^l y^{l-1} + b^l), \tag{1}$$

where W^l is the weight matrix of the first layer, b^l is the paranoid vector of the first layer and ϕ is the ReLU activation function. The meta-learner is a DNN, which produces the classification results. It is connected with the last layers of the low detectors. Its shape is similar to the first-level neural networks, except for the last layer that includes only one neuron and produces the final classification. The final output layer uses softmax function regression to convert the results calculated by neural network forward propagation into a probability distribution. Table 1 shows the neurons used in each layer. Finally, a dropout rate of 0.2 is considered to prevent the generation of overfitting.

TABLE 1
The Neural Network Architecture

Neural Networks	Layer	Neurons
First-level	Input	4000
	Hidden	2000
	Hidden	1000
Second-level	Input	500
	Hidden	100
	Hidden	10
	Output	1

4 Malware Analysis by Multiple Detectors

The exploited features to derive the dynamic behaviour of malware are related to the following observations:

- *Observation 1*: Malware usually invokes critical APIs to perform malicious activities on sensitive data. Basically, malicious software tends to use a greater number of critical API calls than benign apps;
- *Observation 2*: Within the same family, variants of the malware invoke critical API calls by following similar patterns, even if evasive techniques have been adopted to introduce noise in the code execution (e.g., moving, replacing, and adding useless instructions and loops within the code segments);
- *Observation 3*: The use of hardware features could reduce the deficiencies of software features in addressing the malware evasion problem during the dynamic analysis;
- *Observation 4*: During malware execution, malicious traffic is significantly more frequent or wide-spreading than normal traffic, and more than 70 percent of malicious traffic is focused either in the first 5 minutes or is generated when a specific event is triggered or user activity is performed.

According to the previous considerations, the API calls invoked by the malware, its memory dump, and the generated traffic represent the main characteristics adopted by the works presented in the literature and used to extract the malicious behaviour of malware during execution. Therefore, all these features are exploited and correlated to capture different behavioural aspects of malware, which are combined in the form of feature fusion to improve detection performance.

The considered features are extracted by using the Cuckoo sandbox, The analysed samples are executed and the invoked API call sequences are collected and recorded. The '*ma* command' is used to extract the dump data file, which includes the entire virtual address space of the monitored process. Finally, a connection to the monitored host is created by exploiting a hotspot on a mirroring server. Tcpdump is used to capture the network traffic from the phone. In the next paragraph, the five main detection algorithms proposed in the

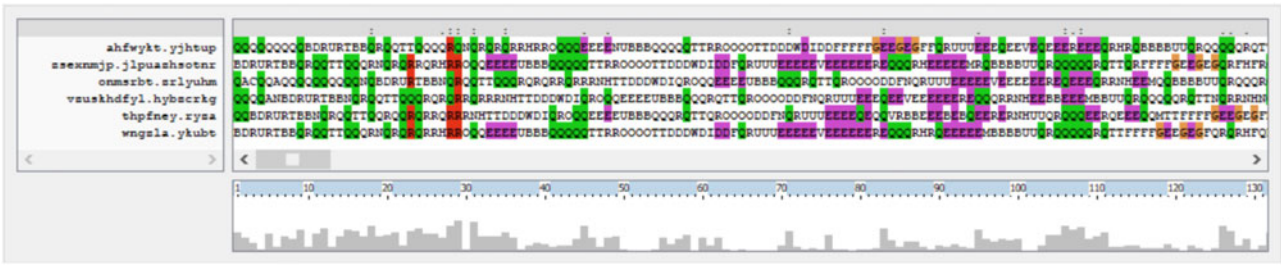

Fig. 2. API call sequences before applying MSA.

literature have been compared and implemented to extract the considered features of the malware behaviour.

4.1 API Call Frequency Detector (AFD)

AFD performs API call frequency analysis to list out the API calls that are more frequent in the malware than in the benign set [14]. The API calls invoked by the malware during its execution are represented as a tree, where the root node is the APK, and the leaves are the API calls. The edges represent the number of occurrences of the specific API call. To capture malicious behaviours only a selected API list is considered. They are identified manually using the Android Developer reference pages [30], as well as the APIs introduced in [31]. Then further refine has been performed, including only those with the largest difference in frequency between malware and benign apps.

4.2 API Sequence Alignment Detector (ASD)

Several works presented in the literature adopted the sequence alignment algorithms, which are widely used in the bioinformatics field to calculate the similarity between two DNA sequences or find certain DNA subsequences from full DNA [32], [33]. They provide alignment solutions to match the maximum common sequence of two DNA sequences, which are slightly different by some noisy tokens. In particular, these algorithms fall into two categories, 'global alignment' and 'local alignment'. The former aligns entire sequences. It is useful to compare sequences of most similar lengths and strings. The Needleman-Wunsch is a well-known global algorithm. The local alignment finds similar subsequences between two sequences of different lengths. An example is Smith-Waterman algorithm. The multiple sequence alignment algorithm (MSA) represents an extension of the previous pairwise alignment, handling multiple sequences at the same time [32]. It can be exploited to identify regions of similarity among API sequences of malware belonging to the same family, by deleting or inserting a defined gap into each sequence [14]. The extracted sequences are the feature vectors representative of the malware behaviours of that specific family. In this work, the ClustalX is adopted for implementing sequence alignment [34]. Moreover, since the MSA algorithm can use only character sequences, the collected sequences of API calls are converted into the character strings, and then, extracted the common API call sequences of each malware family previously clustered. Figs. 2 and 3 show the API call sequences produced by some malware samples belonging to the Ransomware family, respectively before and after applying MSA. In the figures, the horizontal lines represent the API call sequences of malware, whereas the vertical lines represent the API call subsequences common among malware belong to the same family.

4.3 Markov Chain Detector (MCD)

The sequence of state transitions performed by the analysed app is represented as a Markov chain, which is widely used in the literature [16], [35]. According to the Markov chain model, the sequence of API calls can be represented as a graph, in which each node represents the API, whereas the edges connecting one node to another represent the API invoked by that node. Moreover, each edge is labelled with the probability of that transition.

Specifically, the set of states of the Markov chain is represented by all possible API calls invoked by the considered app, and denoted as C. The edge between c_i and c_j denotes the probability P_{ij} that after the API i is invoked the API j

$$P_{ij} = \frac{O(c_i - > c_j)}{\sum_{c_z=1}^{c_z=n} O(c_i - > c_z)}, \tag{2}$$

where $O(c_i - > c_j)$ represents the number of occurrences of the API call c_j after the API call c_i, divided by all the occurrences O_{iz} for all API calls z in the chain. Finally, the sum of the probabilities associated with all edges from any node is equal to 1

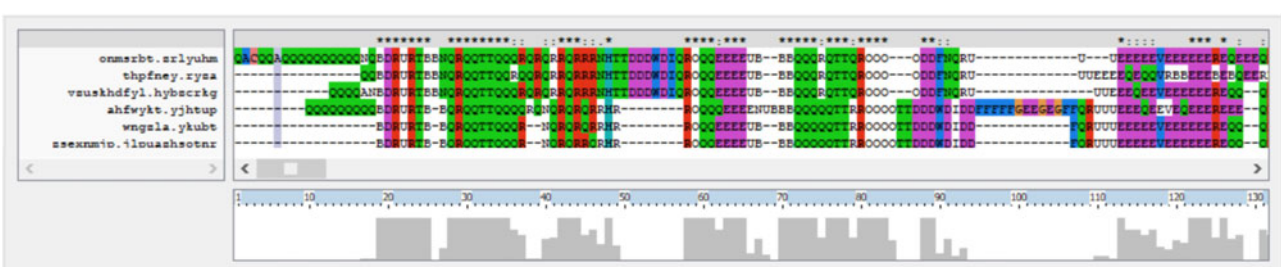

Fig. 3. API call sequences after applying MSA.

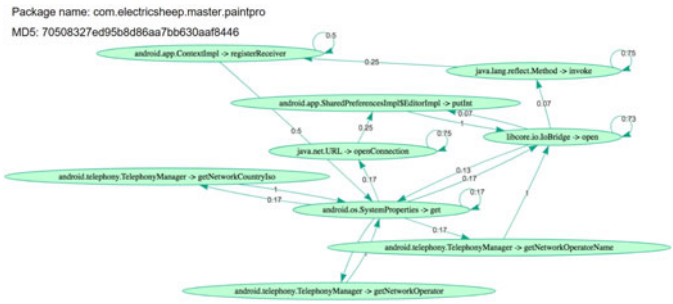

Fig. 4. Markov chain of the com.electricsheep.master.paintpro APK.

$$\sum_{c_z=1}^{c_z=n} P_{iz} = 1. \tag{3}$$

Therefore, the application feature vector is represented by the probabilities of transitioning from one API to another in the chain. Fig. 4 shows an example of the Markov chain of the *com.electricsheep.master.paintpro* APK malware.

4.4 Memory Dump Detector (MDD)

This technique consists in monitoring the malware process memory, and convert the memory data dump binary files into grayscale images, which are compress based on the bicubic interpolation method and used for extracting the malware feature vector [18]. It refers to data in the actual physical memory space and the virtual memory space data. The collected dump binary files (usually bigger than 3MB) is converted in a two-dimensional array with a constant width (2048 pixel) and a height that changes with the size of the dump file. The value range of each element in this matrix is [0, 255] (i.e., 0 is black and 255 is white). Fig. 5 shows the grayscale images of three malware belonging to different families, whereas Fig. 6 represents three malware belonging to the same family.

Then, bicubic interpolation is used to scale the grayscale image, in order to extract feature vectors with equal numerical values. It considers 4*4 pixels according to the following equation:

$$f(x,y) = \sum_{i=0}^{i=3} \sum_{j=0}^{j=3} b_{i,j} X^i y^j, \tag{4}$$

where $f(x,y)$ are continuous derivatives within the unit area, which are spliced together through the bicubic curved surfaces, in order to guarantee that the images are compressed to the same pixel size (the derivatives and the boundaries are matched) [18]. Then, the grayscale image feature is extracted using a histogram of gradient (HOG), adopted for detecting objects in computer image processing. The HOG features are extracted by calculating the total gradient magnitude and direction of the image

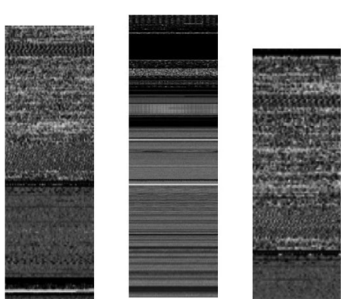

Fig. 5. Memory dump of three malware belonging to different families.

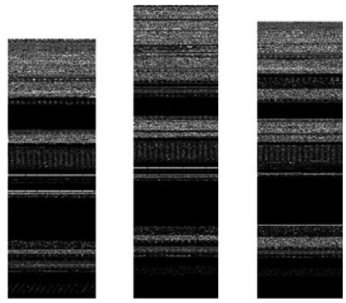

Fig. 6. Memory dump of three malware belonging to the same family.

$$Grad(x,y) = \sqrt{G_x(x,y)^2 + G_y(x,y)^2}, \tag{5}$$

where $Grad(x,y)$ is the gradient magnitude, and

$$g(x,y) = tan^{-1} \frac{G_x(x,y)}{G_y(x,y)}, \tag{6}$$

is the gradient direction, with G_x gradient component in direction x and G_y gradient component in direction y.

Then, the contrast of the HOG cell unit is normalized with a scale for local contrast statistic of one block with the size of 16*16 (composed of 2*2 cells). The moving step length of the window is 8 pixels. The feature vectors of every cell are cascaded based on the voting results, and the vector feature normalization of the block is produced by L2-norm

$$n' = \frac{n}{\sqrt{||n||_2 + e}}, \tag{7}$$

where $||n||_2$ is the L2 norm of feature vector n and e is a not zero constant. Finally, the HOG feature vector is equal to

$$f = (n'_1, n'_2, \ldots, n'_n). \tag{8}$$

4.5 Network Traffic Detector (NTD)

The NTD detector is adopted to analyse the network traffic generated by apps at the network access point. It analyses HTTP requests, TCP and DNS flows to detect malicious network behaviours. The attack behaviour is monitored and filtered by a proxy and stored in different *pcap* format files according to the protocol type. Then, each traffic file is processed to generate the feature vectors.

Fig. 7 shows the proportion of different traffic packets in the application layer. The figure shows that in malicious network traffic, the HTTP traffic is comparable with the encrypted traffic, whereas in benign traffic the proportion of the HTTP protocol is the largest. Moreover, as regard HTTP traffic of benign apps, the request packet size is usually small and the returning data is typically larger, whereas the malicious traffic shows an opposite behaviour. Finally,

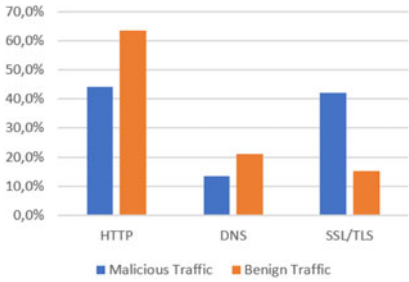

Fig. 7. Malicious and benign traffic.

in the TCP traffic, there are several traffic patterns common in malware belonging to the same family.

As suggested in [20], the following statistical characteristics of the TCP flows have been collected: 1) uploading and downloading packet number, 2) average uploading and downloading packet bytes, and 3) the total uploading and downloading packet number in a session. The considered HTTP request feature set is: 1) the string keys included in the request URI source, 2) the host and port number of the requested resource, 3) the HTTP method, and 4) the user agent information.

5 Performance Evaluation

In order to define which are the combinations of low detectors that can be used to support the unpredictability of the detection strategy, as well as reduce the number of detectors to correlate, the best performing low detectors should be selected. Therefore, the approach consists in: (i) identifying the best performing specialized detectors for each malware type, (ii) comparing the selected specialized detectors against each generic detector with respect to the same malware type that the specialized detectors have been designed for, (iii) selecting the best performing combinations of low detectors (including specialized and generic detectors), and (iv) evaluating the performance rate variation of the final ensemble-detector within the observation time window.

During the evaluation process of the ensemble-detector, different experiments have been performed. Specifically, it is first evaluated by using a validation dataset, which includes samples belonging to the same malware families used during the training process, and then, the effectiveness of the ensemble-detector in detecting unknown malware families not included in the training dataset is verified. Moreover, in order to verify how the presented approach is robust with respect to the evolution over time of malware, the evaluation is first performed with an older malware dataset (between 2013 and 2014) and then, with a more recent dataset (between 2015 and 2017). Moreover, in order to evaluate the robustness of ensemble-detector against the anti-emulation and anti-debugger techniques used by malware developers to evade the detection, the testing dataset is executed on a real smartphone. Finally, the experimental results show how the use of longer observation windows and the application of different detector combinations during the analysis process can improve the detection accuracy and speed, as well as increase the unpredictability of the detection strategy.

TABLE 2
Malware Dataset

Malware family	Number of samples	Malware family	Number of samples
Adrd	87	Hamob	26
AnserverBot	187	Iconosys	133
BaseBridge	352	Imlog	41
Boxer	25	Jifake	27
DroidDream	84	Kmin	112
DroidKungFu	770	MobileTx	67
DroidDreamLight	46	GinMaster	332
ExploitLinuxLotoor	66	Opfake	595
FakeDoc	131	Opfake	595
FakeInstaller	902	Plankton	481
FakeRun	58	Pjapps	58
Gappusin	44	SMSreg	38
Geinimi	107	SendPay	58
Glodream	85	Yzhc	45

5.1 The Initial Dateset

The initial dateset used to identify the best performing combinations of low detectors is composed of different malware families, which has been collected from the Drebin dataset (Arp. 2014), including 5.56k malware samples grouped in 179 different families, and by Malgenome that includes other 1.26k samples grouped into 52 different families. We only considered families with at least 20 samples (as suggested in similar experimental works [17], [36]), as well as removed those samples that were not executed in the Cuckoo sandbox.

To infer precisely the features of each malware family, we categorized the collected malware samples in the belonging family in advance. As Table 2 shows, the final dataset consists of 27 families and 4960 samples. Then, the considered malware families are grouped into malware types, including Backdoor, Worm, and Trojan. Moreover, 1.2K benign apps have also been obtained by Google Play store using the GooglePlayApi tool.

In order to train generic and specialized low detectors, different training data subsets have been used. In particular, since specialized detectors must be designed to detect a specific type of malware, they are trained using only one single malware type at a time, and then evaluated in order to identify the best performing specialized detector for each malware type. Instead, generic detectors are trained using contemporaneously all the malware types. The same set of benign apps have been included in each training dataset. Moreover, in order to compare the identified specialized detectors against each generic detector, the testing dataset included all the malware types that the specialized detectors have been designed for. Therefore, the whole dataset has been divided into three subsets, including training (60 percent), validation (20 percent), and testing (20 percent) that is used for evaluating the performance of the final ensemble-detector.

5.2 Training and Evaluation of Specialized and Generic Low Detectors With Known Malware Families

Since each specialized detector must be designed to detect a specific type of malware, they have been separately trained

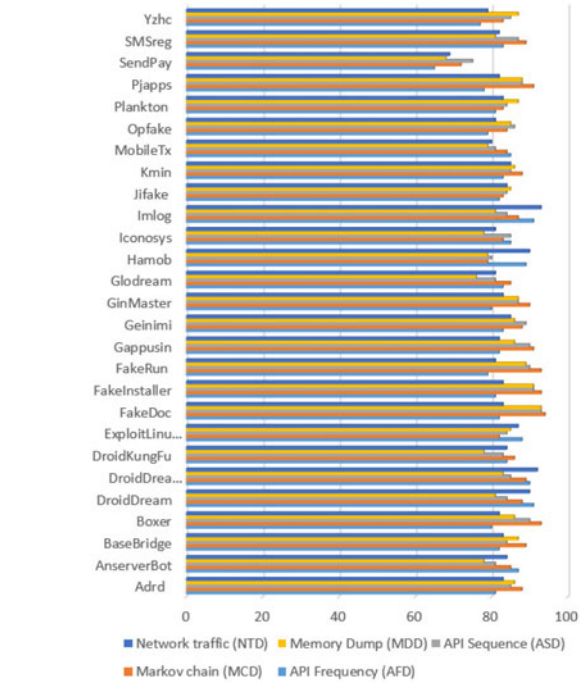

Fig. 8. Area under the curve size of specialized low detectors.

with respect to each type of malware included in the dataset. Then, their performances are compared by using the Receiver Operating Characteristic (ROC) curves. Fig. 8 shows the Area Under the Curve (AUC) achieved by the detectors for each malware type included in the validation dataset.

Results show that AFD detector provides the best performance with respect to Backdoor and Worm malware, ASD detector with Worm, MCD detector with Backdoor and Trojan, MDD detector with Trojan, and NTD detector with Worm and Trojan. Then, each generic detector is trained using the whole validation dataset. Fig. 9 shows the comparison of the generic detectors by ROC curves. The two best performing generic detectors are the Markov chain and API sequence detectors.

5.3 Comparing Specialized and Generic Low Detectors With Respect to Malware Families Included in the Training Dataset

The high performing generic detectors (GDs) have been compared with the best performing specialized detectors (SDs). They are separately tested with respect to each malware type used to train specialized detectors. As Fig. 10 shows, although the Markov chain and API sequence generic detectors provided the best detection performance with respect to the other generic detectors, results confirm the consideration in [4], that is, specialized detectors outperform generic detectors in classifying specific malware types with improvements between 3 to 9 percent.

5.4 Building Ensemble Detector

The last step consists in identifying what mix of low detectors should be combined to build the ensemble-detector. Specifically, in order to select how many and which low detectors have to be selected, the decision function is

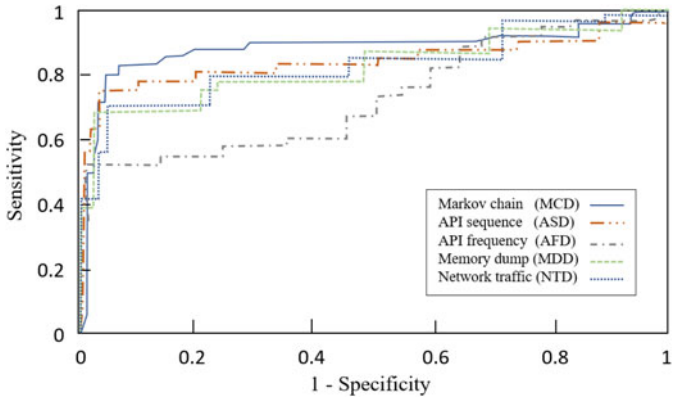

Fig. 9. ROC curves of generic low detectors.

separately tested with different sets of low detectors. The first set (called specialized-ensemble detectors) consists of the best performing specialized detectors, one for each malware type, including the Markov chain detector for Backdoor, API sequence detector for Worm, and Memory Dump for Trojan. The second set of detectors includes one or more generic detectors combined with the best performing specialized detectors (called combined-ensemble detectors). The first-level of low detectors is tested by using as input the validation dataset. The second-level detector is validated by using the output of low detectors. The malware classification is the output of the meta-learner detector.

Regarding the set of specialized detectors, assuming the sensitivity and specificity to be equally important, we have chosen how the best threshold for each detector, the closest point on the ROC curve to the point nearest to the top left corner of the graph. Table 3 shows the threshold values for the set of specialized detectors, as well as the high

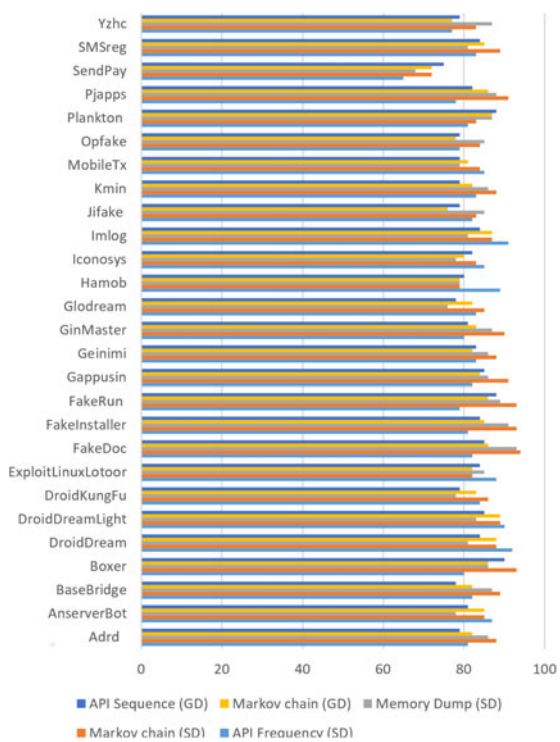

Fig. 10. Comparing area under the curve size of generic detectors with respect to specialized detectors.

TABLE 3
ROC Curve Thresholds for Specialized-Ensemble Detectors

Specialized Detector	Sensitivity Threshold	Confidence Threshold
MCD (Backdoor)	0.781	0.890
MDD (Trojan)	0.756	0.911
ASD (Worm)	0.806	0.872

confidence decision function that maximizes the sensitivity (minimizes the false positives). For the second-level detector, the threshold is equal to 0.612.

For the combined-ensemble detector, several mixes of low detectors have been tested. In particular, combinations of different generic detectors and several specialized detectors have been evaluated. Experimental results have shown that the best combination of low detectors is achieved with Markov chain generic detector and three specialized detectors, including Markov Chain, Memory Dump, and API sequence. Table 4 shows the thresholds chosen for low detectors, whereas for the second-level detector the threshold is fixed to 0.553.

Table 5 shows the comparison in terms of sensitivity, specificity, and accuracy between the specialized-ensemble detector and the best combined-ensemble detectors. Results confirm that a specialized-ensemble detector outperforms combined-ensemble detectors.

5.5 Comparing Specialized and Combined Ensemble Detectors With Respect to Unknown Malware Families not Included in the Training Dataset

In order to test the implemented ensemble detectors with malware types not considered during the training of low detectors, another validation dataset has been used, which includes also Spyware, Keyloggers, Ransomware, Spam, Adware, etc., downloaded by VirusShare dataset [37]. Specifically, the exploited VirusShare dataset includes 24K APKs, dating back from June 2013 to March 2014. Moreover, about 8k benign apps have been collected from Playdrone dataset [38].

Table 6 represents the performance produced by different combinations of generic detectors with the best performing specialized detectors. Results show that the presence of at least one generic low detector can improve the detection rate when the malware to be analysed does not belong to the same families that the specialized detectors have been designed for. Indeed, the analysis of malware families not included in the training dataset entails a drastic reduction

TABLE 4
ROC Curve Thresholds for the Combined-Ensemble Detector

Combined Detector	Sensitivity Threshold	Confidence Threshold
Markov chain (GD)	0.909	0.713
Markov chain (SD)	0.781	0.890
Memory dump (SD)	0.756	0.911
API sequence (SD)	0.806	0.872

TABLE 5
Specialized-Ensemble Detector *versus* Combined-Ensemble Detector

Ensemble-detector	Sens.	Spec.	Accur.	FN%	FP%
Specialized detector	98.7%	100%	98.9%	1.3	0
Combined detector	95.1%	96.9%	95.4%	4.9	3.1

of ensemble detector performance, in the case where it is only implemented by specialized low detectors (the top row of Table 6). Specifically, the ensemble detectors that combine generic detectors with specialized detectors provide a comparable, or even better performance, in particular in terms of sensitivity.

5.6 Evaluating Ensemble Detectors With Malware Running on the Real Smartphone

As described in Section 5.1, the samples that were not executed in the Cuckoo sandbox were removed from the dataset used for training and validation of ensemble-detectors. The reason for the non-execution of such samples was due in part to possible incompatibilities (i.e, they required older vulnerable versions of the mobile platform) or corrupt files, but several samples were sophisticated malware designed to evade dynamic analysis. Malware developers could delay and hinder the malicious behaviour of malware using different evasion techniques. For example, they could wait for a command-and-control server to issue commands over the internet. During start-up, malware could check the environment in which they are executed, and automatically not active the malicious behaviour or simply crash or shut down when they realize to be executed in a sandbox or a virtual environment. Moreover, by using anti-debugger mechanisms, malware could execute unexpected behaviours after detecting the presence of a debugger.

Therefore, in order to evaluate the capability of the proposed detector to detect such elusive advanced malware, the malware dataset is re-executed in a real Android smartphone. However, since many samples are incompatible with respect to the adopted smartphone, as well as running the application on a real smartphone requires several minutes (due to the rebooting and recovery of the device for each execution), only 1.2k samples are considered, equally distributed among the different malware families, and 0.8k benign apps. Moreover, 137 samples that did not run in the sandbox (which are correctly executed in the real smartphone) have been added to the dataset.

In order to monitor the malware behaviour on the real device, Hooker is adopted for the dynamic analysis of Android apps [39]. It can be used to automatically intercept any API calls invoked by the targeted application and rebooting the mobile device. Moreover, Nyuki Android Process Dumper is a user-mode software application that runs nativity on Android devices. It has been used to dump all memory of the running malware [40].

As Table 7 shows, using malware features collected directly on the real device allows improving the detection rate of all ensemble detectors. Moreover, in order to minimize the attack effects to the network and the user devices, the network architecture for the experiment set up has been implemented

TABLE 6
Comparing of Combined-Ensemble Detectors With Respect to Unknown Malware Families

Generic Low Detectors			Specialized Low Detectors			Ensemble-Detector Performance				
MCD	ASD	MDD	MCD	ASD	MDD	Sensitivity	Specificity	Accuracy	FN%	FP%
			X	X	X	80.4%	99.1%	85.1%	19.6	0.9
X			X	X	X	81.9%	98.6%	86.1%	18.1	1.4
	X		X	X	X	81.1%	97.8%	85.3%	18.9	2.2
		X	X	X	X	80.3%	97.4%	84.6%	19.7	2.6
X	X		X	X	X	83.0%	98.7%	86.9%	17.0	1.3
	X	X	X	X	X	80.7%	97.8%	85.0%	19.3	2.2
X		X	X	X	X	81.9%	98.6%	86.1%	18.1	1.4
X	X	X	X	X	X	83.4%	98.9%	87.3%	16.6	1.1
X	X	X				79.8%	97.3%	84.2%	20.2	2.7

TABLE 7
Comparing of Ensemble Detectors With Malware Run on a Real Smartphone

Generic Low Detectors			Specialized Low Detectors			Ensemble Detector Performance				
MCD	ASD	MDD	MCD	ASD	MDD	Sensitivity	Specificity	Accuracy	FN%	FP%
			X	X	X	81.3%	94.5%	84.6%	18.7	5.5
X			X	X	X	83.4%	93.4%	85.9%	16.8	6.6
X	X	X	X	X	X	85.1%	92.9%	87.0%	14.9	7.1
X	X	X				82.6%	92.7%	85.1%	17.4	7.3

according to the configuration presented in [41]. Therefore, some malware that ware able to recognize the presence of the emulated environment when they ran in the sandbox (avoiding to activate the malicious behaviour) are detected.

Overall results have shown that the sensitivity increases, whereas the specificity decreases. Moreover, the sensitivity of generic-ensemble detector (only composed of generic low detectors) improves about twice the detection rate of other combine-ensemble detectors. Probably, the reason is because the evasion actions used by malware to analyse the execution environment are common to a large number of malware and independent of malware type.

5.7 Comparing Detectors With More Recent Dataset

This section explores whether detectors trained on the set of malware presented in Table 2 continue to be effective over time as malware evolve. Therefore, a more recent CICAndMal2017 dataset is used, which includes about 426 malware collected from several sources and 5000 benign apps collected from Googleplay market, published in 2015, 2016, and 2017 [42]. The malware are classified into four categories, such as Adware, Ransomware, Scareware, and SMS Malware.

Table 8 shows that the detection rate of specialized-ensemble detector degrades substantially more than combined-ensemble detectors. This is probably due to the fact that malware over time become more and more evolved and combine the characteristics of different malware types, evading the detection capabilities of specialized detectors if not appropriately retrained.

5.8 Training Time for Detection Model Updating

Although the combined-ensemble is more resilient to malware evolution with respect to a specialized-ensemble, the model should be periodically retrained. This involves updating each specialized detector designed to detect the malware type included in the new dataset, all the generic detectors, and the second-level network. If a new detector specialized for a new malware type has to be added, the second-level must also be retrained. Finally, if a low detector needs to be replaced only the second-level must be retrained. Table 9 shows the training time when (i) the ensemble detector needs to be fully retrained, and (ii) an additional low detector is added. In the first case, although the training of the first-level detectors could be performed in parallel, the table shows the sum of the training times of each first-level network and the second-level network of two specific configurations. For the second case, model updating only requires the training of a single first-level network and the second-level network. The

TABLE 8
Comparing of Ensemble-Detectors With More Recent Dataset

Generic Low Detectors			Specialized Low Detectors			Ensemble Detector Performance				
MCD	ASD	MDD	MCD	ASD	MDD	Sensitivity	Specificity	Accuracy	FN%	FP%
			X	X	X	65.2%	93.6%	72.2%	34.8	6.4
X			X	X	X	72.1%	91.1%	76.8%	27.9	8.9
X	X	X	X	X	X	74.1%	92.9%	78.7%	25.9	7.1
X	X	X				73.2%	88.2%	76.9%	26.8	11.8

TABLE 9
Training Time Evaluation

Case	Detectors			
I	Ensemble Detector			
	1,837 s			
	Generic Low Detectors		Specialized Low Detectors	
II	MCD	607 s	MCD	343 s
	AFD	533 s	AFD	319 s
	ASD	689 s	ASD	384 s
	MDD	576 s	MDD	345 s
	NTD	415 s	NTD	293 s

ensemble detector runs on a workstation with 2.60 GHz Intel Core (TM) i7-9750H, Geforce RTX 3080, and 128 GB RAM.

6 Dynamic Analysis Duration

As discusses by several works presented in the literature, the features extracted during dynamic analysis can highly depend on the duration of the observation window [5], [22]. In this direction, during the analysis on the physical device, we observed that most of the malware generally active the malicious behaviour during start-up, and it is terminated within a few hundred milliseconds. For this reason, the dynamic analysis process of several works presented in the literature lasting only a few seconds or minutes, in order to reduce the detection computation and time [22]. On the other hand, if there are no particular restrictions on the malware execution, intuitively, the attacker could evade detection by advancing the attack very slowly whereas running a mostly normal program [5]. Moreover, many malware trigger the malicious activity by responding to an event (e.g., a specific user-activity) or when a certain date or time is reached. This could make difficult to observe malicious behaviours that go beyond the bootstrapping process unless the program is completely executed. For example, from a network behaviour point of view, post-bootstrap behaviour is often more interesting than what happens in the first few minutes [43]. A malware developer could specifically implement a malicious behaviour so that it only activates after a certain minimum time of application execution. For example, some malware families start after a time delay or communicate to the server periodically, such as MobiDash and Simplocker respectively. Therefore, a longer analysis period could involve a better understanding of malware behaviour. On the other hand, if the malicious activity lasts very little time (a few hundreds of instructions) compared to all the rest of program execution (i.e., the observation window), it could reduce the performance of several detection methods. For example, the number of malicious network messages or API calls can become insignificant with respect to normal behaviour. Moreover, the longer the observation period, the higher the risk that malware can complete its malicious action before being detected and blocked.

A possible solution could consist in splitting out the analysis process within different observation windows, whose sizes must be appropriately chosen according to the possible time intervals in which the malicious actions could be activated during execution. In this direction, each sample is executed for at least one hour. We observed that more than 80 percent of the samples performed malicious behaviours immediately after start-up, and another 8 percent within the first 5 minutes of analysis. The remaining samples activated malicious behaviours after 5 minutes or performed other malicious activities that they have not performed in the first minutes. For example, we observed that in many samples only a minor fraction of all malicious network flows started within the first seconds (about 20 percent) and no more than 70 percent in the first 5 minutes. This result confirms that a longer analysis window can be helpful for a better understanding of malware behaviour, as well as differentiate the observation windows (the weight of collected features in time) could improve the detection rate.

Therefore, in order to detect malware as soon as possible and prevent malicious action from being completed, we performed several experiments by varying the computation period 'T' (i.e., the number of consecutive API calls used to compute the detection function), instead of using a full observation window. Each experiment consists of consecutive measurements, in which the features are collected during an analysis period of T, and then, classified using the detector. In particular, in order to classify the analysed samples, we adopted an approach based on a count-and-threshold mechanism, the so-called *alpha-count*. It relies on a simple α filtering function defined as follows:

$$\begin{cases} \alpha(0) = 0, \\ \alpha(i) = \alpha(i-1) + A & if\ it\ is\ classified\ as\ malware, \\ \alpha(i) = \alpha(i-1) * R & if\ it\ is\ classified\ as\ benign, \end{cases} \tag{9}$$

where i is a discrete time variable, α is a score that keeps track of the decision, and $R \in]0, 1[$. Malware is detected if $\alpha(i)$ exceeds a certain threshold α_T, which represents the minimum number of consecutive measurements necessary for classifying the app as malware. R represents the ratio in which $\alpha(i)$ is decreased for each classification as benign, whereas A is the increment for each classification as malicious. The values to assign to α_T, A, and R depend on the accuracy required. In particular, a higher value of α_T can reduce the false positives (increasing the precision); on the other hand, it can reduce the accuracy and increase detection delay. A lower value of R can reduce the false negatives (i.e., elusive malware that scatter the malicious behaviour in the source code in order to avoid long sequences of malicious code that could be more easily detected by patter marching and API call sequence analysis), but increase the false positives. Moreover, the values of α_T and R must be chosen on the base of the computation period T. During the experimental evaluation, we observed that too small computation periods cause a high number of false positives, which decreases significantly with larger periods. This was particularly evident during the first observation windows, especially in those ensemble detectors that included API-based detector. This was probably due to the presence of frequent behaviours (i.e., short API call sequences) common to both malware and benign during start-up. On the other hand, if the window grew excessively, false negatives also increased, i.e., short malicious behaviours could not be detected. Therefore, a good compromise to balance the two phenomena consists in not exceeding 10000 API calls (i.e., included in the range

TABLE 10
Comparing Combined-Ensemble Detectors With Respect to Three Observation Windows
(With R = 0.750 and A = 0.35 During the Start-Up; R = 0.988 and A = 1 After the Start-Up)

Observer.	Generic Low Detectors				Specialized Low Detectors		Ensemble Detector Performance				
Window	MCD	ASD	MDD	NTD	MCD	ASD	Sensitivity	Specificity	Accuracy	FN%	FP%
	X				X	X	86.13%	96.27%	88.63%	13.87	3.73
		X			X	X	85.87%	95.85%	88.34%	14.13	4.15
			X		X	X	85.34%	95.51%	87.85%	14.66	4.49
				X	X	X	83.17%	95.13%	86.12%	16.83	4.87
I	X	X	X	X	X	X	88.96%	99.81%	91.64%	11.04	0.19
	X	X	X		X	X	89.27%	98.73%	91.61%	10.73	1.27
	X	X		X	X	X	88.61%	98.76%	91.12%	11.39	1.24
	X		X	X	X	X	88.34%	98.02%	90.73%	11.66	1.98
		X	X	X	X	X	87.43%	97.84%	90.00%	12.57	2.16
	X				X	X	88.03%	96.70%	90.17%	11.97	3.30
		X			X	X	87.50%	96.24%	89.66%	12.50	3.76
			X		X	X	87.28%	95.83%	89.39%	12.72	4.17
				X	X	X	85.00%	95.44%	87.58%	15.00	4.56
II	X	X	X	X	X	X	91.46%	99.96%	93.56%	8.54	0.04
	X	X	X		X	X	91.41%	99.19%	93.33%	8.59	0.81
	X	X		X	X	X	91.13%	99.25%	93.14%	8.87	0.75
	X		X	X	X	X	89.61%	98.43%	91.79%	10.39	1.57
		X	X	X	X	X	88.44%	98.28%	90.87%	11.56	1.72
	X				X	X	89.76%	96.71%	91.48%	11.24	3.29
		X			X	X	88.64%	96.12%	90.49%	11.36	3.88
			X		X	X	88.33%	96.03%	90.23%	11.67	3.97
				X	X	X	87.91%	95.24%	90.72%	12.09	4.76
III	X	X	X	X	X	X	94.18%	99.98%	95.61%	5.82	0.02
	X	X	X		X	X	91.52%	99.31%	93.44%	8.48	0.69
	X	X		X	X	X	93.72%	99.26%	95.09%	6.28	0.74
	X		X	X	X	X	92.04%	98.54%	93.65%	7.96	1.46
		X	X	X	X	X	90.90%	98.42%	92.76%	9.10	1.58

[1k:10k]). Moreover, regarding the distribution of malicious code, it was observed that many malware concentrated all their malicious actions immediately after start-up, or performed short but frequent actions during the first seconds. In the rest of the execution, the malicious actions were very short (a few tens or hundreds of instructions), more discontinuous and distributed. Therefore, a higher T can imply lower α_T and R values.

For one hour of analysis, we observed that a good compromise was to divide the analysis process into three separated time observation windows: the start-up (the first 5 seconds), the subsequent 5 minutes of observation, and the last window that included the remaining time. Moreover, the best performance was obtained applying the *alpha-count*-mechanism separately in the three windows. The used dataset was the same exploited in Section 5.5. We performed different experiments varying the combination of parameters T, A R, α_T for each observation window. We observed that R and A must be lower during the first observation window (start-up) than the other two windows. Table 10 shows the detection performance achieved in the three observation windows, by using the two best specialized low detectors and different combinations of generic low detectors. The results have shown that the use of a larger observation time and consecutive computation periods can improve the detection performance. Specifically, the overall accuracy was increased more than 4 percent (after 5 minutes) with respect to the combined-ensemble detectors without the *alpha-count*-mechanism (Table 6). Moreover, although the ensemble detector that included both MCD, ASD, and MDD generic low detectors presented the best sensitivity during the start-up, all the configurations that included NTD produced a higher percentage increase in sensitivity at the end of the analysis process with respect to the results achieved at the end of the second observation window. In particular, the configuration that included MCD, ASD, and MDD generic low detectors provided an increase of at least 2 percent less with respect to the configurations that included NTD.

Fig. 11 shows the distribution of detection time of two different combinations of low detectors, during an observation window of 1 hour. According to the achieved results, during the first observation window (the first 5 s) about 87.74 percent of the malware were detected by the ensemble detector that

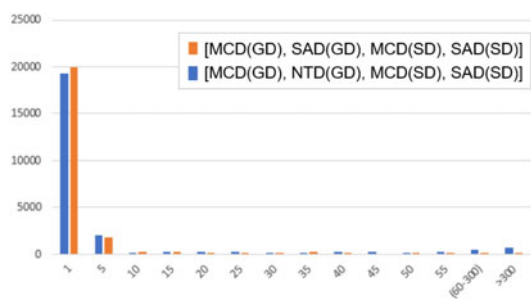

Fig. 11. Comparing detection time (s).

TABLE 11
Comparing of Combined-Ensemble Detectors With Respect to Unknown Malware Families

Ensemble Detectors	Low Detectors				Sensitivity	Specificity	Accuracy	FN%	FP%
Specialized	MCD (SD)	ASD (SD)	MDD (SD)	NTD (SD)	85.71%	98.47%	88.86%	14.29	1.53
Alpha-count	MCD (GD)	NTD (GD)	MCD (SD)	ASD (SD)	92.06%	97.01%	93.28%	7.94	2.99

included MCD and NTD generic low detectors, with a Mean-Time-To-Detect (MTTD) of 0.201 s. Using an observation window of 5 minutes, 88.98 percent of samples were detected with an MTTD of 2.547 s. After one hour, about 91.89 percent of the malware were detected. The maximum detection time was 57 minutes, with an MTTD of 27.91 s. The ensemble detector that included API-based generic low detectors (MCD and ASD) produced an MTTD of 0.146 s and sensitivity of 89.13 percent during the first observation window. Moreover, it offered a marked MTTD improvement (2.071 s) at the end of the third observation window, but with a lower sensitivity (equal to 90.01 percent).

This proves that the detector performance varies according to the observation window. Therefore, using different combinations of detectors during the analysis process, the detection performance and speed could be further improved. For example, the use of API-based generic detectors during the first two windows and a network-based generic detector after 5 minutes involved a sensitivity improvement with a clear reduction in the MTTD, which was 1.133 s at the end of the second observation window (with a sensitivity greater than 90 percent) and 21.71 s after 1 hour (with a sensitivity of 92.26 percent).

6.1 Comparison With Other Ensemble Detectors

Table 11 compares the performance of the specialized-ensemble detector that includes the four best specialized low detectors with respect to the *alpha-count*-ensemble that includes two generic and two specialized low detectors. Results confirm that the *alpha-count*-ensemble detector outperforms specialized-ensemble detectors, in particular in terms of sensitivity and accuracy (about 4.42 percent). The number of false positives also decreases significantly, although it still remains slightly higher than the specialized-ensemble detector.

Table 12 compares the *alpha-count*-ensemble with the mixed-ensemble detector presented in [4], which includes the same number of low detectors, but with a different composition: one generic and three specialized detectors. The results show that the mixed-ensemble provides greater sensitivity, but a significant reduction in specificity and accuracy compared to the *alpha-count*-ensemble. However, the higher sensitivity may be due to the use of a larger number of specialized low detectors. Moreover, it is not clear whether the testing dataset used in [4] also included malware families not belonging to the same types that the specialized detectors had been trained.

TABLE 12
Alpha-Count-Ensemble Versus Other Multi-Features Detectors

Ensemble Detectors	Sensitivity	Specificity	Accuracy
Mixed-ensemble [4]	94.4%	89.8%	91.7%
MNN-Static [23]	99%	NA	98%
Deep-Detector [44]	84%	NA	90%
Ensemble AT+DAE [45]	97.54 %	97.53 %	97.54%
Alpha-count	92.06%	97.01%	93.28%

Table 12 also shows the comparison with some works proposed in the literature that combined multiple features or detectors. According to the presented results, the proposed techniques produced an MTTD less than 2.5 s with generally higher accuracy values. In particular, Kim *et al.* [23] presented an ensemble detector that exploited different static features, with an accuracy equal to 98 percent. However, they used more low-level neural networks (five, one for each feature). Moreover, a perfect comparison cannot be made considering that, in almost all works the observation window is smaller than 5 minutes, as well as the testing set used by the considered works belonged to the same dataset used for the training. The *alpha-count*-ensemble detector has been trained with datasets (Malgenome and Drebin) completely different from the testing set (VirusShare), which contained samples that did not belong to the same families used for training. Indeed, using the same dataset (VirusShare), the *alpha-count*-ensemble reached 99.3 percent with 2 generic and 3 specialized detectors.

7 Conclusion and Final Remarks

The retraining of deterministic ensemble classifiers is a time-consuming operation that may be useless, as several works have proved [5]. Malware developers could reverse-engineer the retrained detector, or however, evade it again. Stochastic switching between different detectors, in a way that cannot be predicted by the attacker, could make more complex the implementation of such evasion strategies [29]. Moreover, the resilience of the detection technique increases with the number of diversities introduced. Therefore, starting from these assumptions, this paper presents an extensive experiment campaign that shows the detection performance as a function of the incorporated low detectors and the observation window, useful to implement multiple strategies to optimally combine detectors (exploring different combinations of specialized and generic low detectors, detection algorithms, and *alpha-count* settings). Moreover, the possibility of varying the length of the observation window and switching between different combinations of detectors during the analysis process allows the implementation of new schemes needed to increase the unpredictability of the detection strategy. Technically this would require the training of a second-level network for each detector combination, whereas the first-level networks would remain unchanged.

The experimental results have confirmed how the combined-ensemble detectors provide a comparable or even better performance with respect to specialized-ensemble (in particular in terms of sensitivity) with unknown malware families not included in the training dataset. The comparison

of ensemble-detectors with a more recent dataset proves that the detection rate of specialized-ensemble degrades substantially more than combined-ensemble if not appropriately retrained, involving a more frequent update. Moreover, the execution in the real device allows collecting some behaviours that are not collected in a simulated environment, resulting in an increase of sensitivity of generic-ensemble detector, about twice the detection rate of the other combined-ensemble detectors. Finally, the results have proved that the use of an *alpha-count*-mechanism with larger observation time windows can increase the detection accuracy. Furthermore, switching detectors during the analysis process can further improve detection sensitivity and speed, as well as offer further schemes for combining detectors, useful to increase the detection strategy unpredictability.

In future work, exploiting recent results in Probably Approximately Correct learnability theory cloud be measured the resilience of the different schemes against the evasion techniques. Finally, a secure framework should be built to allow the different schemes to be periodically updated as malware evolve, as well as automatically switched during the detection process.

ACKNOWLEDGMENTS

This work was supported by Università degli Studi della Campania Luigi Vanvitelli (id: 10.13039/501100009448).

REFERENCES

[1] Kaspersky lab., "Detects 360,000 new malicious files daily up 11.5% from December 2017." Accessed: Jul. 2019. [Online]. Available: https://www.kaspersky.com/about/press-releases/2017_kaspersky-lab-detects-360000-new-malicious-files-daily/

[2] AV-TEST Inst, "Malware statistics & trends report." Accessed: Jul. 2020. [Online]. Available: https://www.av-test.org/en/statistics/malware/

[3] Chet Hosmer, "Polymorphic & metamorphic malware." Accessed: Jul. 2019. [Online]. Available: https://www.blackhat.com/presentations/bh-usa-08/Hosmer/BH_US_08_Hosmer_Polymorphic_Malware.pdf

[4] K. N. Khasawneh, M. Ozsoy, C. Donovick, N. Abu-Ghazaleh , and D. Ponomarev, "EnsembleHMD: Accurate hardware malware detectors with specialized ensemble classifiers," *IEEE Trans. Dependable Secure Comput.*, vol. 17, no. 3, pp. 620–633, May/Jun. 2020.

[5] K. N. Khasawneh, N. Abu-Ghazaleh , D. Ponomarev, and L. Yu, "RHMD: Evasion-resilient hardware malware detectors," in *Proc. 50th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2017, pp. 315–327.

[6] G. D'Angelo, M. Ficco, and F. Palmieri, "Association rule-based malware classification using common subsequences of API calls," *Appl. Soft Comput. J.*, vol. 105, pp. 1–8, Jul. 2021,

[7] Y. Aafer, W. Du, and H. Yin, "DroidAPIMiner: Mining API-level features for robust malware detection in android," in *Proc. Int. Conf. Secur. Privacy Commun. Netw.*, 2013, pp. 86–103.

[8] E. Mariconti, L. Onwuzurike, P. Andriotis, E. De Cristofaro , G. Ross, and G. Stringhini, "MAMADROID: Detecting android malware by building Markov chains of behavioral models," in *Proc. 24th Netw. Distrib. Syst. Secur. Symp.*, 2017, pp. 1–22.

[9] Y. Qiao, Y. Yang, J. He, C. Tang, and Z. Liu, "CBM: Free, automatic malware analysis framework using API call sequences," in *Proc. Adv. Intell. Syst. Comput.*, vol. 214, 2014, pp. 225–236.

[10] Y. Ki, E. Kim, and H. K. Kim, "A novel approach to detect malware based on API call sequence analysis," *Int. J. Distrib. Sensor Netw.*, vol. 11, no. 6, 2015, Art. no. 659101.

[11] H. Kim, W. Khoo, and P. Li, "Polymorphic attacks against sequence based software birthmarks," in *Proc. 2nd ACM SIGPLAN Workshop Softw. Secur. Protection*, 2012, pp. 1–8.

[12] C. In Kyeom, K. TaeGuen, S. Y. Jin, P. Haeryong, C. Bomin, and I. G. Eul, "Malware similarity analysis using API sequence alignments," *J. Internet Serv. Inf. Secur.*, vol. 4, no. 4, pp. 103–114, 2014.

[13] A. Elhadi, M. Maarof, andB. Barry, "Improving the detection of malware behavior using simplified data dependent API call graph," *Int. J. Secur. Appl.*, vol. 7, no. 5, pp. 29–42, 2013.

[14] K. Hyunjoo, K. Jonghyun, K. Youngsoo, K. Ikkyun, K. J. Kuinam, and K. Hyuncheol, "Improvement of malware detection and classification using API call sequence alignment and visualization," *Cluster Comput.*, vol. 22, pp. 921–929, 2019.

[15] M. Ficco, "Comparing API call sequence algorithms for malware detection," in *Proc. Adv. Intell. Syst. Comput.*, 2020, pp. 847–856.

[16] M. Ficco, "Detecting IoT malware by Markov chain behavioral models," in *Proc. IEEE Int. Conf. Cloud Eng.*, 2019, pp. 229–234.

[17] A. Martín, V. Rodríguez-Fernández, and D. Camacho, "CANDYMAN: Classifying android malware families by modelling dynamic traces with Markov chains," *Eng. Appl. Artif. Intell.*, vol. 74, pp. 121–133, 2018.

[18] Y. Dai, H. Li, Y. Qian, and X. Lu, "A malware classification method based on memory dump grayscale image," *Digit. Invest.*, vol. 27, pp. 30–37, 2018.

[19] M. Ozsoy, K. N. Khasawneh, C. Donovick, I. Gorelik, N. Abu-Ghazaleh , and D. Ponomarev, "Hardware-based malware detection using low-level architectural features," *IEEE Trans. Comput.*, vol. 65, no. 11, pp. 3332–3344, Nov. 2016.

[20] S. Wanga, Z. Chena, Q. Yanb, and Z. Jiaa, "Mobile malware detection method using behavior features in network traffic," *J. Netw. Comput. Appl.*, vol. 133, pp. 15–25, 2019.

[21] L. Tenenboim-Chekina, A. O. Barad, D. M. Shabtai, B. Shapira, and Y. Elovici, "Detecting application update attack on mobile devices through network features," in *Proc. IEEE Conf. Comput. Commun. Workshops*, 2013, pp. 91–92.

[22] Z. Chen *et al.*, "First look at android malware traffic in first few minutes," in *Proc. IEEE Trustcom/BigDataSE/ISPA*, 2015, pp. 206–213.

[23] T. Kim, B. Kang, M. Rho, S. Sezer, and E. Gyu Im , "A multimodal deep learning method for android malware detection using various features," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 3, pp. 773–788, Mar. 2019.

[24] M. N. A. Zabidi, M. A. Maarof, and A. Zainal, "Malware analysis with multiple features," in *Proc. 14th Int. Conf. Comput. Modelling Simul.*, 2012, pp. 231–235.

[25] A. Mohaisen, O. Alrawi, and M. Mohaisen, "AMAL: High-fidelity, behavior-based automated malware analysis and classification," *Comput. Secur.*, vol. 52, pp. 251–266, 2015.

[26] R. S. Pirscoveanu, S. S. Hansen, T. M. T. Larsen, M. Stevanovic, J. M. Pedersen, and A. Czech, "Analysis of malware behavior: Type classification using machine learning," in *Proc. Int. Conf. Cyber Situational Awareness Data Analytics Assessment*, 2015, pp. 1–7.

[27] M. Ozdemir and I. Sogukpinar, "An android malware detection architecture based on ensemble learning," *Trans. Mach. Learn. Artif. Intell.*, vol. 2, no. 3, pp. 90–106, 2014.

[28] C. Smutz and A. Stavrou, "When a tree falls: Using diversity in ensemble classifiers to identify evasion in malware detectors," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2016, pp. 1–15.

[29] Y. Vorobeychik and B. Li, "Optimal randomized classification in adversarial settings," in *Proc. Int. Conf. Auton. Agents Multi-Agent Syst.*, 2014, pp. 485–492.

[30] Android Developer Reference Page. Feb. 2019. [Online.] Available: https://developer.android.com/reference/packages.html

[31] A. Yousra, W. Du, and H. Yin, "DroidAPIMiner: Mining API-level features for robust malware detection in Android," in *Proc. Int. Conf. Secur. Privacy Commun. Syst.*, 2013, pp. 86–103.

[32] Multiple sequence alignment. Accessed: Feb. 2019. [Online]. Available: http://www.ebi.ac.uk/Tools/msa/

[33] G. D'Angelo, M. Ficco, and F. Palmieri, "Malware detection in mobile environments based on Autoencoders and API-images," *J. Parallel Distrib. Comput.*, vol. 137, pp. 26–33, 2020.

[34] ClustalX, sequence alignment tool. Accessed: Feb. 2019. [Online]. Available: http://www.clustal.org/

[35] G. Fink, *Markov Models for Pattern Recognition: From Theory to Applications*. Berlin, Germany: Springer, 2014.

[36] S. K. Dash *et al.*, "DroidScribe: Classifying android malware based on runtime behavior," in *Proc. IEEE Security Privacy Workshops*, 2016, pp. 252–261.

[37] Virusshare malware dataset. Dec. 2018. [Online]. Available: https://virusshare.com/
[38] Playdrone goodware dataset. Accessed: Mar. 2018. [Online]. Available: https://archive.org/details/playdrone-apks
[39] Hooker - An opensource tool for dynamic analysis of Android apps. Accessed: Jan. 2019. [Online]. Available: https://github.com/AndroidHooker/hooker
[40] Nyuki Android process dumper (AProcDump)–A user-mode software application capables of acquiring the volatile memory of processes. [Online]. Available: https://www.silensec.com//downloads-menu/software/aprocdump
[41] A. H. Lashkari, A. F. A. Kadir, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark Android malware datasets and classification," in *Proc. Int. Carnahan Conf. Secur. Technol.*, 2018, pp. 1–7.
[42] Android malware dataset (CICAndMal2017 - First part). [Online]. Available: https://www.unb.ca/cic/datasets/andmal2017.html
[43] C. Rossow, C. J. Dietrich, H. Bos, L. Cavallaro, F. C. Freiling, and N. Pohlmann, "Sandnet: Network traffic analysis of malicious software," in *Proc. 1st Work. Building Anal. Datasets Gathering Experience Returns Secur.*, 2011, pp. 78–88.
[44] N. Mchaughlin *et al.*, "Deep android malware detection," in *Proc. ACM Conf. Data Appl. Secur. Privacy*, 2017, pp. 301–308.
[45] D. Li, Q. Li, Y. Ye, and S. Xu, "A framework for enhancing deep neural networks against adversarial malware," *IEEE Trans. Netw. Sci. Eng.*, vol. 8, no. 1, pp. 736–750, First Quarter 2021.

Massimo Ficco received the PhD degree in computer engineering from the University of Naples "Parthenope", Naples, Italy. He is an associate professor with the University of Campania "Luigi Vanvitelli", Italy. His research interests include cloud security, malware analysis, and software reliability of critical infrastructures. He serves as the editor-in-chief, board-editor, and reviewer of several international journals, and has been conference chair and member of many international conference committees. He is co-founder of WAYSafety s.r.l. a spin-off company working in development of IoT-Based systems.