Future Generation Computer Systems 148 (2023) 93–105

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Privacy-preserving malware detection in Android-based IoT devices through federated Markov chains

Gianni D'Angelo, Eslam Farsimadan, Massimo Ficco, Francesco Palmieri *, Antonio Robustelli

Department of Computer Science, University of Salerno, Via Giovanni Paolo II, 132, I-84084, Fisciano (SA), Italy

ARTICLE INFO

Article history:
Received 14 January 2023
Received in revised form 21 April 2023
Accepted 20 May 2023
Available online 26 May 2023

Keywords:
Malware classification
Federated learning
Markov chains
Privacy

ABSTRACT

The continuous emergence of new and sophisticated malware specifically targeting Android-based Internet of Things devices is causing significant security hazards and is consequently fostering the need for effective detection models and strategies able to work with these hardware-constrained devices. In addition, since such models are often trained on confidential application data, many involved subjects are reluctant to share their data for this purpose. Accordingly, several Federated Learning-based solutions are emerging, which rely on the capabilities of Machine Learning models in malware detection/classification without sharing user data. However, Federated Learning methods are often adversely affected by non-independent and identically distributed data in terms of both the required training time and classification results. Therefore, a promising solution could be to overcome the Federated Learning-related issues by preserving the privacy of end-user data. In this direction, the capabilities of Markov chains and associative rules are extended within a federated environment to face malware classification tasks in the IoT scenario. The presented approach, evaluated on several malware families, has achieved an average accuracy of 99% in the presence of centralized and decentralized unbalanced training/testing data by overcoming the most common state-of-the-art approaches. Also, its runtime performance is comparable with centralized ones by considering several non-independent and identically distributed dataset partitions, splitting criteria, and clients, respectively.

1. Introduction

The exponential growth of the Internet of Things (IoT) technology, together with the success of the Android OS, caused the explosion of the number of mobile apps developed for many kinds of devices empowered by such OS, such as smart TVs, smart watches, refrigerators and other interconnected gadgets that can be easily controlled by using common smartphones. Also, since such smartphones frequently assume the role of IoT gateways for many mission-critical applications, they prove to be one of the main driving forces within IoT ecosystems. However, despite their fundamental role in the IoT forefront, such devices introduce new cybersecurity issues and risks. In particular, due to the lack of appropriate protection mechanisms on the most famous Android-based embedded platforms, the large volume of yearly released malware applications specifically targeted for these environments poses challenges that foster the introduction of detection and classification strategies that are reliable and effective against the different malware families [1].

According to earlier classification studies [2–4], the intensive usage of concealment and obfuscation strategies is one of the primary factors related to the significant growth of malware targeting IoT devices. Therefore, to tackle the fast development and evolution of malware in the IoT environment, it is crucial to design robust and reliable malware detection and classification strategies that are sensitive and effective against these obfuscation techniques. However, malware applications usually belong to families characterized by similar behaviors, implying that most new malware is derived as new versions of already existing malware. Hence, the possibility of developing strategies that can effectively categorize malware depending on its family, regardless of being a variant, appears particularly promising to prevent and control its evolution over time. For this reason, many dynamical analysis strategies, based on the observation of the application behavior described through sequences of system-level API calls, have been proposed. More precisely, these strategies assume that malicious applications might contain a set of well-distinct APIs invoked more often or in a different order than those called by genuine ones [5].

* Corresponding author.
E-mail addresses: giadangelo@unisa.it (G. D'Angelo), efarsimadan@unisa.it (E. Farsimadan), mficco@unisa.it (M. Ficco), fpalmieri@unisa.it (F. Palmieri), arobustelli@unisa.it (A. Robustelli).

https://doi.org/10.1016/j.future.2023.05.021

For instance, Markov chains are one of the most effective cutting-edge strategies for describing the API calls invoked by applications and constructing the representative behavioral patterns of particular malware families [6]. They consider the sequence of API calls to model the application-related behavior as a graph in which each node represents a unique API, while each edge represents the transition probability between two APIs. Also, Markov chains-based detectors have been proven resistant to evasion efforts against irrelevant API calls injected throughout malicious code and thus be effective against polymorphic malware [7].

However, building a sufficiently complete knowledge base on emerging malware applications is currently a slow and challenging process also for state-of-the-art Machine Learning (ML) and Deep Learning (DL)-based solutions. In addition, the involved organizations (companies and end-users) are often unwilling to share their data because they are focused on preserving their own privacy as well as the intellectual property often related to their IoT applications and systems.

Alternatively, Federated Learning (FL)-based approaches represent a recent privacy-preserving solution, which leverages ML and DL models' capabilities to face several classification and detection tasks without sharing data [8–10]. However, as highlighted in many literature studies [11–14], non-Independent and Identically Distributed (non-IID) data often adversely affects FL-based models regarding the required training time, convergence, learning processes, and classification results. Also, such strategies are often strongly influenced by the configuration of some additional hyperparameters (e.g. threshold values) that might limit their applicability.

Aimed by this motivation, we present a federated Markov chains-based paradigm for malware detection in Android-based IoT scenarios. Such a paradigm makes data owners proactive contributors to the related building process, also giving them a mechanism to timely update the global model without sharing their private raw data (e.g., the execution history or installed applications). To this purpose, Markov chains and associative rules are exploited within a federated logic, in which users independently process the applications execution data and then send the extracted information to a central server primarily dedicated to setting up and sharing the proposed detector. Next, we analyze the effectiveness of the proposed strategy, compared with famous ML-based approaches, within a realistic IoT scenario in which we used a dataset of around 3500 malware belonging to 8 Android families. Finally, we evaluate the required time effort by considering several dataset partitions and involved clients. Therefore, we show that the proposed federated architecture can obtain comparable time performances in the presence of non-IID data.

Hence, the main contributions of this paper can be summarized as follows:

1. A federated architecture is presented to support the rules mining process by leveraging Markov chains;
2. The resulting associative rules-based detector is used for recognizing the different malware families by considering both centralized and decentralized data;
3. A performance study is done to show the effectiveness of the proposed approach in the presence of non-IID data.

The remainder of the paper is structured as follows. Section 2 will present the related works about malware classifiers based on Markov chains and federated models, respectively. Section 3 will report a background overview of the presented associative rules-based detector. Section 4 will describe the proposed Federated architecture. Finally, Section 5 will discuss the experimental results, while Section 6 will show the conclusions and future work.

2. Related works

Most of the existing malware are designed in order to evade the pattern-matching detection mechanisms. Such kinds of applications are also known as polymorphic and metamorphic malware and can mutate the appearance of their code by adding several NOP and loop instructions, permuting user registers, and modifying static data structures. For this reason, several dynamic-based methods have been proposed in the last years [2,15–17]. However, because most of them are characterized by a considerable computational complexity (i.e., they are often NP-Complete) [18,19], several studies have adopted Markov chain-based models that represent the applications-related features (e.g., the invoked API Calls pairs) as corresponding transition probabilities. In addition, such approaches have also shown their effectiveness against evasion techniques by proving that the insertion and removal of arbitrary calls do not significantly affect the transition probabilities.

For instance, in 2018, Martin et al. [7] presented a tool that classifies Android malware families by arranging the extracted dynamic features as a Markov chain. The following application, also known as CANDYMAN, has been validated on a collection of 4.442 samples by achieving a precision of 81.8%. In 2019, Ficco [6] proposed an IoT anomaly detector by considering the transaction probabilities associated with the API call sequences invoked by an application. In detail, the following model has obtained an F-measure of 89% on 22.000 benign applications and 24.000 malware collected from several famous datasets. Instead, in 2021, D'Angelo et al. [3] presented a new malware classifier realized by combining the Markov chains and associative rules. More precisely, they have proven the effectiveness of the proposed solution on 8 famous malware families by achieving an average F-score of 96.1%.

On the other hand, due to the recent explosion of IoT-related malware applications and the necessity of protecting privacy, the involved users and organizations became increasingly reluctant to share their data. For this reason, one of the most popular and recent options is associated with Federated Learning (FL)-based solutions [20], in which each entity trains an individual model using only its data and, in order to create a global and shared model, sends the derived model parameters to a central server [21]. In this direction, many FL-based solutions have been proposed and adopted in several application domains, such as Healthcare applications [22], Failure prognosis [23], and Network traffic detection and classification [24,25]. Also, they have been used for Malware classification in IoT environments by considering several extracted features, FL algorithms optimization, learning models, and security schemes. Unfortunately, due to the high amount of related works published, performing a complete comparison is very difficult. A detailed comparison among the different solutions proposed in the literature is presented [25–27].

In 2020, Ruei-Hau et al. [9] proposed a new Android detection schema to prevent possible Poisoning attacks on a federated model. They protected the federated learning process through a Secure Multi-Party Computation (SMPC) implementation provided by OpenMined. Moreover, they evaluated the discussed detection model in centralized and decentralized scenarios by obtaining an average accuracy of 94.05% and 93.45%, respectively. In the same direction, Galvez et al. [8] presented Less is More (LiM), a Semi-supervised framework that leverages FL and static features to detect Android malware applications. More precisely, the following framework achieved an average F-Score of 95% over 50 iteration rounds and 50.000 Android applications distributed among 200 clients. In 2021, Shukla et al. [28] proposed a Robust and Active Protection with Intelligent Defense (RAPID)

strategy against malicious activities based on CNNs. They have proven the effectiveness of the following malware classifier by obtaining a 94% average accuracy. Also, to mitigate possible global model-related poisoning, they introduced a server-side defense mechanism based on the Euclidean distances derived from each federated model. Finally, in 2022, Rey et al. [10] and Popoola et al. [29] presented similar network flows-based approaches to detect the presence of cyberattacks in IoT domains. More precisely, the following methods have been trained by considering several learning scenarios (Supervised, Semi-Supervised, and Unsupervised), DNNs models, hyperparameters (number of clients and rounds), and network features by achieving a 99% average accuracy.

However, the reported FL-based approaches consider static features that, as previously remarked, are strongly affected by obfuscation techniques and polymorphic malware. Instead, traffic-based ones, since considering features directly derived from the network packets, become ineffective against traffic anonymization techniques. Also, since they cannot analyze the application-related dynamic behavior, they cannot be used in any possible runtime-based detection or classification strategies [17].

Furthermore, as highlighted in many notable literature studies [11–14], FL-based solutions are often adversely affected by non-Independent and Identically Distributed (non-IID) data. Hence, to also overcome such issues, many learning strategies have been proposed in recent years. In 2020, Karimireddy et al. [11] presented the Stochastic Controlled Averaging algorithm (SCAFFOLD) that tries to estimate the update direction for both server and client models. More precisely, the proposed algorithm, validated in the presence of 100 clients and unbalanced dataset partitions, has outperformed the most famous state-of-the-art Federated algorithms by slightly reducing the required time effort. Li et al. proposed [12] an extension of the FedAvg algorithm based, at each iteration, on the selection of a subset of clients through their local loss function values. The presented FedProx algorithm, tested on four famous federated datasets, has achieved comparable IID and non-IID loss functions but at the expense of slower convergence. In 2021, Lu et al. presented [30] an improvement of the FedAvg algorithm based on Earth Mover's Distance (EMD) between central and local parameters. More precisely, a new metric, defined as node degree contribution, is derived to improve the local models-related aggregation at each iteration. Also, the experimental results carried out with 100 clients and a different number of iterations, have shown a similar convergence behavior and a slight improvement in accuracy compared with the FedAvg algorithm. Finally, Paragliola et al. [31] proposed a PartialNet Strategy that reduces communication costs by considering partially trained models that, at each iteration, are aggregated by the central server if and only if they satisfy a given threshold. The following strategy, validated on a real-world dataset regarding people affected by hypertension, has proven its effectiveness by reducing communication costs in both IID and non-IID data scenarios.

However, the proposed FL-based strategies are often characterized by additional hyperparameters whose tuning complexity could limit their applicability [25–27]. Therefore, in order to overcome the discussed issues and preserve the intellectual property and privacy of the involved users and companies, we extend the associative rules-based detector, presented in [3], within a federated logic. More precisely, we accomplish this by leveraging the potential of Markov chains within a federated scenario. Also, we show the effectiveness of the proposed approach in presence of non-IID data that, instead, often adversely affect the convergence and learning processes of the classic FL-based models.

3. Background

Since a sequence of API calls can be effectively used to model the most representative behavioral features associated with a specific malware application, the background concepts related to the association rules-based detector, presented in [3], are recalled in this Section. More precisely, we first report its workflow through a step-by-step example. Then, we provide some mathematical definitions related to the rules pruning phase, which is essential to obtain relevant classification results.

3.1. Association rules-based detector

The execution flow of a specific application t_m can be represented as a sequence of API calls. As a consequence, rules representing the given application can be extracted in the form $\{API_i \rightarrow API_j\}$, with $API_i \prec API_j$. Note that such rules could include API calls not necessarily contiguous. The number of skipped API calls is considered the "spacing" of the rule. After that, such rules can be associated with nodes of a Markov chain to represent the application as a sequence of independent state transitions [7,32]. Ultimately, we can describe such an API calls flow (i.e. a specific application) by using a graph, with nodes representing two (not necessarily contiguous) API calls and edges identifying their transition timeline (the sequence of invocation) [6]. In order to mine any possible insight from the API calls sequence, different pairs of API calls (rules) are extracted by varying the spacing.

Hence, with a little abuse of notation needed for simplification purposes, the related training process consists, for each application t_m, of several progressive steps n, with $k \in [1, \ldots, n]$ representing the spacing in terms of positions to be skipped within the API calls sequence.

For each intermediate step k (with $k < n$), all the $k-1$ spaced transitions between two API calls are arranged in a Markov-like chain represented by a graph $G_k^{t_m}$, in which the x-th node, N_x, is defined as follows:

$$N_x = [(API_i \rightarrow API_j), \sigma_x] \tag{1}$$

where $(API_i \rightarrow API_j)$ represents the mined rule and σ_x is the related number of occurrences, while the edges represent the transition states.

Next, in the last step ($k = n$), all the graphs are merged into one, namely G^{t_m}, representing the entire execution profile of the application t_m. Also, both the edges and the total number of occurrences of each node x of G^{t_m}, i.e. $\sigma^{t_m}(x)$, are updated. In particular, the occurrences are computed by summing the occurrences σ_x of the same nodes of the previous $(n-1)$ graphs.

For instance, let $t_1 = ADDDBCDD$ and $t_2 = ADDBCCCC$ be the API call sequences of two applications, by assuming only three steps (n = 3), Figs. 1 and 2 show the extracted graphs $G_k^{t_m}$ of each analyzed application at time-step $k = 1$ and $k = 2$, respectively. As depicted, at time-step $k = 1$, the transitions considered are contiguous, and no elements are present between two API calls used for mining a rule. Contrary, at the time step $k = 2$, every rule is extracted by skipping an API call.

Next, at time-step $k = 3$, the previous graphs are merged into a new graph G^{t_m} representing the Run-time behavior of a considered application, as shown in Fig. 3.

Finally, since the set of association rules mined from different applications can be used as a signature characterizing a specific malware category (or class) $c \in C$ (where $C = \{c_1, \ldots, c_{|C|}\}$ is the set of malware categories), all the graphs G^{t_m} associated with $\mathcal{N}(c)$ applications belonging to the class c of the training dataset T, are further merged as a final graph G_c, in which the total number of occurrences of each node, σ_c, is again updated as well as the related edges, as shown in Fig. 4. Note that, as

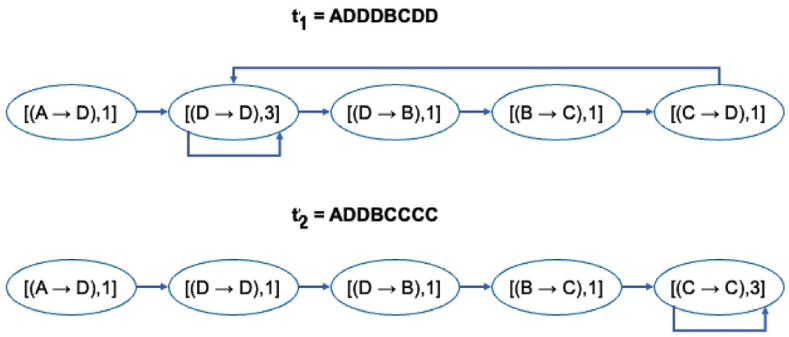

Fig. 1. Extracted graphs $G_k^{t_1}$ and $G_k^{t_2}$ at time-step $k = 1$.

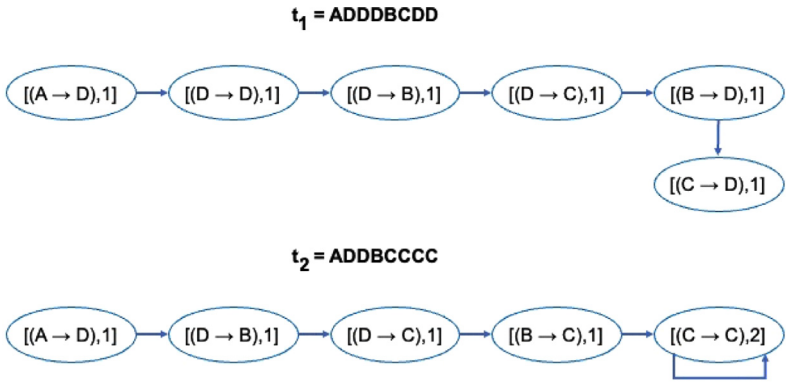

Fig. 2. Extracted graphs $G_k^{t_1}$ and $G_k^{t_2}$ at time-step $k = 2$.

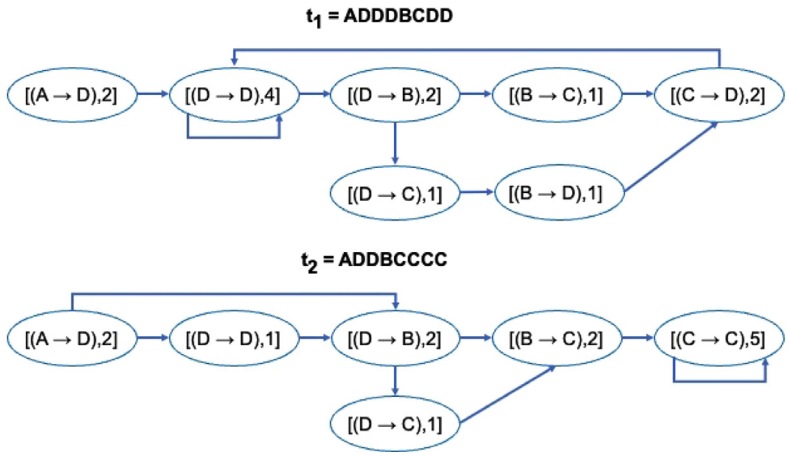

Fig. 3. Merged graphs G^{t_1} and G^{t_2} related to t_1 and t_2 when $k = 3$.

better explained below (see Eq. (3), σ_c is estimated by taking into account the different lengths (in terms of number of API calls) of the applications. Therefore, the proposed detector is characterized by a time complexity of $O(|T| \times n \times l)$, where $|T|$ is the training dataset dimension, n is the number of rules extraction-related steps, and l is the number of API calls.

3.2. Pruning phase definition

Since many rules could be extracted from the training process, a pruning step is of paramount importance to remove unnecessary information and thus achieve relevant classification results. The pruning step is performed by comparing the occurrence of each rule with a threshold value. For instance, Fig. 5 shows all the pruned rules (see dashed nodes) if we consider a threshold less than 2.

However, the existence of similar rules among different malware families could lead to incorrect classification results. To address this issue, a more complex pruning process is needed, in which only rules whose values of support and confidence satisfy a given property (e.g., associated with a specific threshold value) are considered valid.

More precisely, let $A = \{a_1, \ldots, a_{|A|}\}$ the set of admissible API calls, a generic rule is defined as follows:

$$R_{pq} = \{a_p \rightarrow a_q\} \tag{2}$$

with $a_p, a_q \in A$ and $a_p \prec a_q$, which denotes that the API a_p is called before than a_q.

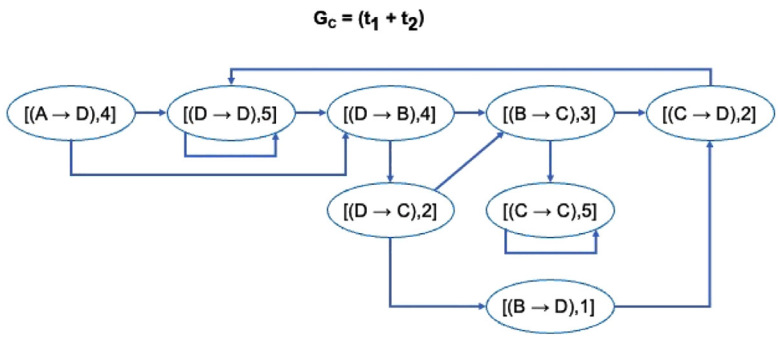

Fig. 4. Final graph G_c derived by merging t_1 and t_2.

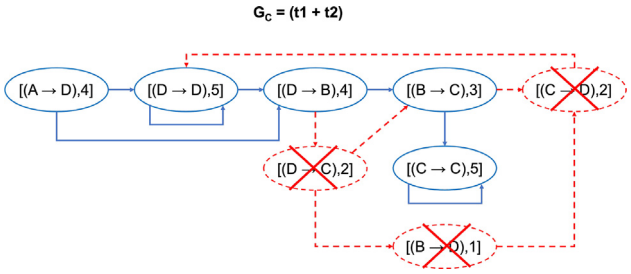

Fig. 5. Pruning phase of rules with $\sigma_x < 2$.

By recalling the aforementioned association between rules and nodes in the merged graph G^{t_i} associated with the application $t_i \in T$, and defining $\sigma^{t_i}(R_{pq})$ as the number of occurrence of the rule R_{pq} in a given application $t_i \in T$, then the support of the rule R_{pq} with respect to the class c is defined as follows:

$$\Phi^c(R_{pq}) = \frac{|R_{pq}| \sum_{i=1}^{\mathcal{N}(c)} \frac{\sigma^{t_i}(R_{pq})}{l_i}}{\mathcal{N}(c)} \tag{3}$$

where l_i is the number of API calls of t_i, while $|R_{pq}|$ is the cardinality of the rule (i.e. 2).

Notice that, to take into account the different APIs flow lengths that could occur among the applications of a given class c as well as the unbalancing among the applications falling within classes of T, the terms l_i, $|R_{pq}|$, and $\mathcal{N}(c)$ of Eq. (3) are used to normalize the support within the range [0, 1].

However, as it is known, the support is not sufficient to estimate the quality of the rules in representing the applications for multi-class contexts. Thus, the confidence of a given rule R_{pq} on a class c is also defined, as follows:

$$\Gamma^c(R_{pq}) = \frac{\Phi^c(R_{pq})}{\sum_{v \in C} \Phi^v(R_{pq})} \tag{4}$$

Eq. (4) expresses the ability of a rule to be unique for a specific class. Indeed, high values denote high uniqueness, while low values indicate that the rule is also present in other malware classes.

As depicted, Eqs. (3) and (4) express a metric characterizing a given rule concerning a class c, and thus, they can be involved in the pruning phase. More specifically, rules having support and confidence less than given thresholds are pruned.

3.3. Classification

Once the training phase is performed and, thus, several rules are obtained, the classification of new incoming applications needs to be implemented. To accomplish this, the following metrics are introduced [3].

Firstly, the following confidence is provided:

$$\Gamma^c_{R_{pq}}(t_m) = \begin{cases} 0 & R_{pq} \not\subseteq t_m, \\ \sigma^{t_m}(R_{pq}) * \Gamma^c(R_{pq}) & R_{pq} \subseteq t_m \ \wedge \ \gamma(t_m) = c, \\ 1/\Gamma^c(R_{pq}) & R_{pq} \subseteq t_m \ \wedge \ \gamma(t_m) \neq c. \end{cases} \tag{5}$$

where $\gamma(t_m)$ is the hypothesis of membership classes associated with t_m.

It represents the confidence of a rule R_{pq} with respect to a class c associated with an application t_m. As shown, its value depends on the presence of the rule R_{pq} within the application t_m, as well as on the value assumed by the hypothesis of membership classes.

After that, the degree of belonging to class c of an application t_m is estimated by evaluating a rank ρ, which is given by:

$$\rho^c(t_m) = \sum_{c \in C} \sum_{\forall p,q} \Gamma^c_{R_{pq}}(t_m), \tag{6}$$

where the summation on p and q takes into account all the rules derived from the training phase.

Finally, the softmax function is used to classify the application t_m, as follows:

$$\gamma(t_m) = \arg\max_{h \in C} \frac{e^{\rho^h(t_m)}}{\sum_{v \in C} e^{\rho^v(t_m)}}. \tag{7}$$

4. The proposed architecture

This section presents the proposed federated architecture. More precisely, we extend the support and confidence indexes within a privacy-preserving federated environment. Then, we provide some implementation details related to core modules. Finally, we describe the proposed schema by highlighting its main advantages.

4.1. Federated indexes definition

The first step necessary for implementing the previously-discussed detector within a federated logic is to extend both support and confidence by considering the entire dataset T split among several federated clients. Note that different clients could deal with similar malware. As a consequence, T could include multiple copies of the same applications.

Let M be the number of considered clients, and T_j be the j-th dataset gathered by the client j, then the entire dataset T is subject to the following:

$$T = \bigcup_{j=1}^{M} T_j \tag{8}$$

We remark that, during the learning process, no T_j dataset is sent to the central server, but only the applications-related graphs

G_c are sent, guaranteeing privacy. Furthermore, Eq. (8) expresses the ability of our approach to face the problem of learning from non-IID data. Notice that this is true because according to Eq. (8), all partial graphs are merged into a single one, and then the support and confidence are evaluated only by the central server.

Accordingly, let $\mathcal{N}(c, T_j)$ be the number of application $t_i \in T_j$ whose class is c, then the federated support of the rule R_{pq} can be defined as follows:

$$\Phi_f^c(R_{pq}) = \frac{|R_{pq}| \sum_{j=1}^{M} \sum_{i=1}^{\mathcal{N}(c,T_j)} \frac{\sigma^{t_i}(R_{pq})}{l_i}}{\mathcal{N}(c)} \tag{9}$$

Similarly, the federated confidence of a given rule R_{pq} on a class c is defined as follows:

$$\Gamma_f^c(R_{pq}) = \frac{\Phi_f^c(R_{pq})}{\sum_{v \in C} \Phi_f^v(R_{pq})} \tag{10}$$

Finally, Eqs. (5) and (6) can be easily generalized to the federated case by replacing Φ with Φ_f and Γ with Γ_f, respectively and, also in this case, the classification of an application is performed by Eq. (7).

4.2. The federated rules-based detector

To overcome the confidentiality and privacy issues related to sharing data, we present an architecture that aims to support malware classification tasks by embedding the proposed associative rules-based detector within a federated logic, in which the involved IoT devices need to send their data to a central aggregation point devoted to sharing information. More precisely, the proposed architecture aims to provide a privacy-preserving workflow in which federated entities asynchronously share only their applications-related graphs (and not the raw data) and receive the malware detector. Thus, the proposed architecture can guarantee the interchange of sensitive information, and their protection against the most common data leakage threats, as done in traditional Federated Learning-based approaches. Furthermore, it also avoids the issues related to integration operations. Indeed, the centralized aggregation of graphs is extremely simple and presents a low computational effort. In addition, since Markov chains are defined as a memoryless stochastic process, each set of mined k-spaced associative rules, derived from the dynamic analysis of API calls, represents a locally trained model, which is equivalent to the local model of the classic FL-based solutions.

Moreover, the proposed workflow needs (in theory) only one centralized aggregation to build the presented classifier, and that does not need any usage of sophisticated algorithms, such as the Federated Averaging (FedAvg) [33], Federated Matched Averaging (FedMA) [34], and Federated Distance (FedDist) [35]. Therefore, the obtained results no longer depend on applications processed by the federated entities (e.g., influenced by particular data distribution) but only from the derived global model. Indeed, since the defined support and confidence indexes (see Eqs. (9) and (10)) are computed on the centralized and aggregated associative rules, we highlight the ability of the proposed logic to overcome the non-IID data-related issues that, instead, adversely affect the traditional federated learning-based solutions.

For this reason, the presented workflow differs from pure FL-based ones and is also extremely suitable to improve the convergence process among Edge and Cloud infrastructures, with specific reference to data aggregation, data security, and services migration [36–38]. Hence, to report as much detailed information as possible, we describe the resulting architecture, structured according to a Publish–Subscribe model/policy, by defining three processes named *Client-Side Extraction*, *Server-Side Aggregation*, and *Detector Update*, respectively.

Therefore, the main goals of the proposed architecture can be summarized as follows:

1. A data extraction workflow is needed to collect associative rules from each federated entity (Client-Side Extraction);
2. A data aggregation workflow is useful to manage the received rules as category graphs and share a malware detector with each entity (Server-Side Aggregation);
3. A data update workflow is necessary for periodically re-adapting and re-sharing the malware detector (Detector Update).

4.2.1. Client-side extraction process

At the beginning of the Client-Side Extraction process, each federated entity asks to subscribe to the central server and receives the number of steps n to run. Next, for each analyzed application, and at each iteration $k < n$, the client extracts the k-spaced associative rules. Finally, the client sends the related graph to the central server. Fig. 6 reports the discussed Client-Side Extraction process, while its steps can be summarized as Alg. 1.

Algorithm 1 Client-Side Extraction

```
Require: T_j (j-th dataset of applications)
n ← askSubscription()
for each t ∈ T_j do
    k ← 1
    G ← ∅
    c ← Class(t)
    apisList ← traceAPIs(t)
    while k < n do
        G ← extractRules(apisList,k)
        k ← k + 1
    end while
    sendGraph(G,c)
end for
```

More precisely, for each application t, Alg. 1 derives the corresponding class c and the list of API calls (*apisList*). Next, at each iteration $k < n$, the algorithm extracts the set of k-spaced associative rules by storing them in G. Note that, after the while loop, G will represent the graph containing any mined rules. Finally, the algorithm sends G and c to the central server and processes another application.

4.2.2. Server-Side Aggregation process

The Server-Side Aggregation process has the fundamental task of collecting the application-related graphs to share the proposed model with each federated entity. To accomplish this, the server first merges each graph with those previously received. Then, it uses the obtained information (i.e. the learned rules) to perform the pruning phase and share the malware detector. Note that the described process confers to the server the capability of continuously sharing an updated classifier each time a new application is received. Therefore, the server can immediately support the federated entities for their detection activities, also in presence of zero-day malware. Fig. 7 shows the Server-Side Aggregation process, while its steps can be summarized as Alg. 2.

Algorithm 2 Server-Side Aggregation

```
Require: M (number of subscribed clients)
for each i ≤ M do
    (G,c) ← receiveGraph(i)
    graphList[c] ← mergeGraph(G)
    detector ← pruningPhase(graphList)
    sendToClients(detector)
end for
```

More precisely, every time the i-th client sends the graph G and the corresponding class c, Alg. 2 merges G with the graphs

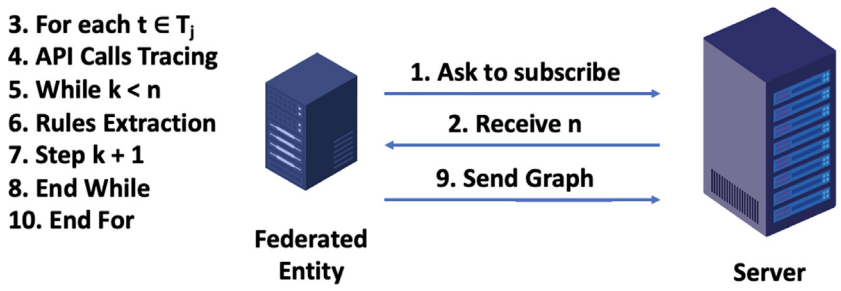

Fig. 6. The high-level steps of the Client-Side Extraction process that summarize the application pre-processing and sending of the related graph.

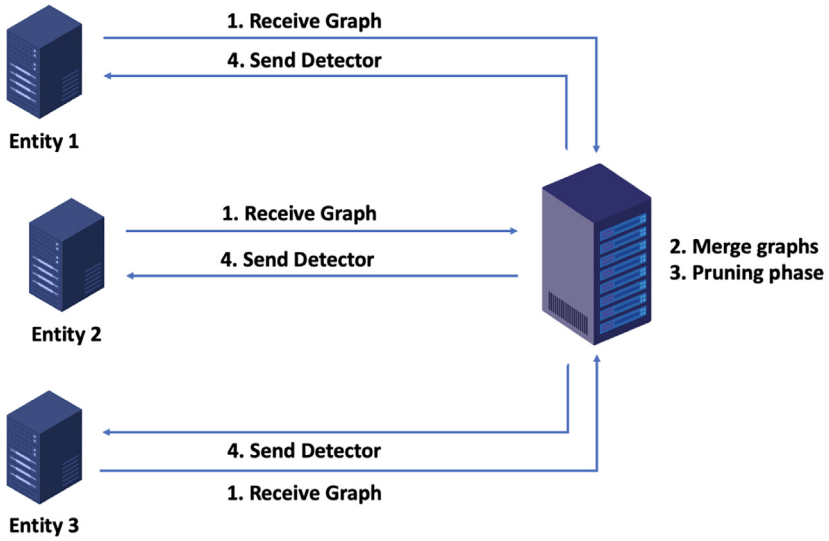

Fig. 7. The high-level steps of the Server-Side Aggregation process that summarize the pruning phase and sharing of the detector.

previously stored in *graphList*[*c*]. Note that *graphList*[*c*] contains the set of *k*-spaced rules related to class *c*. Next, the algorithm performs the pruning phase and sends the *detector* to each subscribed client.

4.2.3. Detector update process

This process is responsible for taking into account new malware applications when a detector has already been shared. It combines the previous phases by considering new applications. We point out the ability of the presented architecture to continuously share an updated classifier regardless of the presence of non-IID data. When an unknown malware application is detected from a federated entity, the application-related graphs are built and sent to the server along with the related malware class. Next, the server will merge the received graphs with those already stored, and after the pruning phase, it will share the updated malware detector with each subscribed entity.

5. Experimental results

The first goal of the experiments is devoted to demonstrating the contribution of the proposed architecture concerning the classification of several malware applications, while the second is to study the required computational effort through performance analysis. To accomplish this, we show the effectiveness of the proposed detector, compared with other state-of-the-art approaches, by considering both centralized and decentralized data. Next, we analyze the required computational effort in the presence of several federated entities and non-IID data partitions, respectively.

Table 1
Dataset division according to the 70/30 criteria.

Family	Training	Testing	Total
Airpush	265	114	379
DroidKungFu	700	301	1001
Fusob	117	49	166
Genpua	220	94	314
GinMaster	372	160	532
Jisut	376	161	537
Opfake	431	184	615
SmsPay	122	52	174
Total	2603	1115	3718

5.1. Dataset and experimental setting

The dataset considered in the following experiments has been derived by Unisa Malware Dataset (UMD) [39], composed of about 3500 applications grouped into 8 Android families: Airpush (Air), DroidKungFu (DKF), Fusob (Fus), Genpua (Gen), GinMaster (Gin), Jisut (Jis), Opfake (Opf), and SmsPay (Sms). More precisely, to evaluate the effectiveness of the proposed detector, we first divided the related API-Calls dataset into training and testing sets according to the 70/30 criteria, as reported in Table 1.

Thus, we used the obtained sets for the centralized and decentralized learning scenarios by respectively considering the following splitting criteria and the number of involved clients:

- Splitting: Horizontal (Samples of each category equally distributed), Vertical (Samples of a category assigned only for some clients), and Mixed;

- Clients: 4, 8, 12, and 16.

To accomplish this, we simulated the proposed architecture within a socket-based Client-Server scenario, using a MacBook Pro equipped with an Apple M1 CPU and 16 GB of unified memory for the server-side and several virtualized clients Linux-based with 2 Processors and 2 GB RAM, respectively.

5.2. Evaluation metrics

To appreciate the classification quality of the proposed detector, we derived the following evaluation metrics from the multi-class confusion matrix: Accuracy (Acc.), Sensitivity (Sens.), Specificity (Spec.), Precision (Prec.), F-Score (F-Mea.), and Area Under the ROC Curve (AUC).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{11}$$

$$Sensitivity = \frac{TP}{TP + FN} \tag{12}$$

$$Specificity = \frac{TN}{TN + FP} \tag{13}$$

$$Precision = \frac{TP}{TP + FP} \tag{14}$$

$$F - Score = \frac{2 * Sens * Prec}{Sens + Prec} \tag{15}$$

$$AUC = \frac{Sens + Spec}{2} \tag{16}$$

For each category, TPs (True Positives) refer to the applications correctly classified, while TNs (True Negatives) refer to the applications correctly identified in another category. Conversely, FPs (False Positives) are the applications mistakenly identified as the considered category, while FNs (False Negatives) present the applications mistakenly identified in another category. Notice that all reported values vary in the range [0, 1]. Also, in order to achieve a global evaluation of the proposed detector, the average performance values (Avg.) and the standard deviation (Dev.) for each considered metric.

5.3. Achieved results

To demonstrate the effectiveness and evaluate the performances of the proposed federated architecture, we first considered the reference dataset as centralized data processed by one client only. More precisely, for each application related to the training set, all the possible associative rules have been mined by using a progressive spacing $k \in [1, 100]$ and then sent to the server as related graphs. Then, for each malware family, we merged the corresponding graphs as one representative graph. Next, since the number of extracted rules was too high and could adversely affect the classification results, we performed the pruning phase by considering any possible value of support (supp.) and confidence (conf.) between 0.1 and 1.0. Finally, we used the testing set to evaluate the quality of the selected training rules by achieving the best classification performances using the combination of supp./conf. = 0.7/1.0. The obtained results, respectively shown in Tables 2 and 3, demonstrate the effectiveness of the detector in classifying several malware families with an average accuracy of 99% and a limited number of FPs and FNs.

Subsequently, to show the effectiveness in a federated environment, we repeated the training process by considering several training set sub-partitions distributed, each time, among different clients. As shown in Table 4, since the application-related graphs extraction process is performed only by clients, the proposed federated detector achieved excellent classification performances independently of the splitting criteria used and the number of subscribed clients, respectively.

Table 2
Confusion matrix related to centralized data.

	Air	DKF	Fus	Gen	Gin	Jis	Opf	Sms
Air	112	0	0	1	1	0	0	0
DKF	4	289	0	2	1	0	0	5
Fus	0	0	49	0	0	0	0	0
Gen	1	0	0	89	1	0	0	3
Gin	1	3	1	1	153	0	0	1
Jis	0	0	0	0	0	161	0	0
Opf	0	0	0	0	0	0	184	0
Sms	1	1	0	6	1	0	0	43

Table 3
Performance results related to centralized data.

Family	Acc.	Sens.	Spec.	Prec.	AUC	F-Mea.
Air	0.9917	0.9412	0.9979	0.9825	0.9696	0.9614
DKF	0.9854	0.9863	0.9851	0.9601	0.9857	0.9731
Fus	0.9991	0.9800	1.0000	1.0000	0.9900	0.9899
Gen	0.9863	0.8990	0.9950	0.9468	0.9470	0.9223
Gin	0.9899	0.9745	0.9925	0.9563	0.9835	0.9653
Jis	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
Opf	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
Sms	0.9836	0.8269	0.9914	0.8269	0.9092	0.8269
Avg.	0.9920	0.9510	0.9952	0.9591	0.9731	0.9549
Dev. (±)	0.0064	0.0566	0.0050	0.0539	0.0291	0.0539

Table 4
Comparison with a different number of clients and splitting criteria (only Avg. values are reported).

Clients	Acc.	Sens.	Spec.	Prec.	AUC	F-Mea.
Centr.	0.9920	0.9510	0.9952	0.9591	0.9731	0.9549
4	0.9920	0.9510	0.9952	0.9591	0.9731	0.9549
8	0.9920	0.9510	0.9952	0.9591	0.9731	0.9549
12	0.9920	0.9510	0.9952	0.9591	0.9731	0.9549
16	0.9920	0.9510	0.9952	0.9591	0.9731	0.9549

Table 5
Performance results related to Random Forest.

Family	Acc.	Sens.	Spec.	Prec.	AUC	F-Mea.
Air	0.9748	0.9023	0.9844	0.8844	0.9434	0.8933
DKF	0.9474	0.9494	0.9468	0.8513	0.9481	0.8977
Fus	0.9998	1.0000	0.9998	0.9969	0.9999	0.9985
Gen	0.9677	0.8063	0.9871	0.8830	0.8967	0.8429
Gin	0.9681	0.8757	0.9869	0.9318	0.9313	0.9029
Jis	0.9988	0.9979	0.9990	0.9934	0.9984	0.9956
Opf	0.9970	0.9900	0.9978	0.9822	0.9939	0.9861
Sms	0.9690	0.6348	0.9902	0.8034	0.8125	0.7093
Avg.	0.9778	0.8946	0.9865	0.9158	0.9405	0.9033
Dev. (±)	0.0177	0.1170	0.0160	0.0672	0.0594	0.0907

5.4. Comparison and discussion

Next, to highlight the potentialities of the presented classifier, we first compared the achieved results with those derived by the most common ML-based methods provided by the Scikit-learn [40] Python library, namely: the Random Forest (RF) algorithm, Linear Support Vector Machine (SVM) classifier, Decision Trees (DTs), and Gaussian Naive Bayes (GNB) classifier, respectively. We have specifically chosen these methods because of their effectiveness in classification runtime performance. To accomplish this, we arranged the k-spaced APIs sequences as corresponding adjacent matrices, in which each APIs pair has been represented with the related frequency. Tables 5–8 report the obtained classification metrics for each ML-based method used, while Table 9 compares these results with those derived by our approach.

As reported in Table 9, the proposed approach outperformed each traditional ML-based method taken into consideration. More

Table 6
Performance results related to Linear SVM.

Family	Acc.	Sens.	Spec.	Prec.	AUC	F-Mea.
Air	0.9682	0.8820	0.9797	0.8517	0.9308	0.8666
DKF	0.9422	0.9042	0.9544	0.8642	0.9293	0.8837
Fus	0.9973	1.0000	0.9971	0.9585	0.9986	0.9788
Gen	0.9557	0.7907	0.9756	0.7960	0.8832	0.7933
Gin	0.9556	0.8578	0.9755	0.8770	0.9167	0.8673
Jis	0.9978	0.9901	0.9989	0.9929	0.9945	0.9915
Opf	0.9931	0.9736	0.9955	0.9639	0.9846	0.9688
Sms	0.9525	0.4810	0.9824	0.6346	0.7317	0.5472
Avg.	0.9703	0.8599	0.9824	0.8674	0.9212	0.8622
Dev. (±)	0.0211	0.1582	0.0139	0.1078	0.0813	0.1352

Table 7
Performance results related to Decision Trees.

Family	Acc.	Sens.	Spec.	Prec.	AUC	F-Mea.
Air	0.9516	0.8116	0.9702	0.7828	0.8909	0.7969
DKF	0.9336	0.8715	0.9535	0.8574	0.9125	0.8644
Fus	0.9986	1.0000	0.9985	0.9786	0.9993	0.9892
Gen	0.9461	0.7297	0.9721	0.7591	0.8509	0.7441
Gin	0.9489	0.8473	0.9696	0.8501	0.9084	0.8487
Jis	0.9982	0.9979	0.9983	0.9888	0.9981	0.9933
Opf	0.9936	0.9843	0.9948	0.9586	0.9895	0.9713
Sms	0.9519	0.5201	0.9792	0.6135	0.7496	0.5629
Avg.	0.9653	0.8453	0.9795	0.8486	0.9124	0.8464
Dev. (±)	0.0250	0.1530	0.0153	0.1206	0.0803	0.1375

Table 8
Performance results related to Gaussian NB.

Family	Acc.	Sens.	Spec.	Prec.	AUC	F-Mea.
Air	0.9432	0.7546	0.9682	0.7587	0.8614	0.7567
DKF	0.8760	0.7551	0.9149	0.7401	0.8350	0.7475
Fus	0.9925	1.0000	0.9920	0.8935	0.9960	0.9438
Gen	0.9449	0.6280	0.9830	0.8165	0.8055	0.7100
Gin	0.9033	0.5411	0.9771	0.8280	0.7591	0.6545
Jis	0.9850	0.9994	0.9828	0.8984	0.9911	0.9462
Opf	0.9359	0.9811	0.9304	0.6334	0.9558	0.7698
Sms	0.9541	0.5894	0.9772	0.6215	0.7833	0.6050
Avg.	0.9419	0.7811	0.9657	0.7738	0.8734	0.7667
Dev. (±)	0.0361	0.1784	0.0259	0.0993	0.0888	0.1151

Table 9
Comparison with ML-based methods (only Avg. values are reported).

Method	Acc.	Sens.	Spec.	Prec.	AUC	F-Mea.
Proposed	0.9920	0.9510	0.9952	0.9591	0.9731	0.9549
RF	0.9778	0.8946	0.9865	0.9158	0.9405	0.9033
SVM	0.9703	0.8599	0.9824	0.8674	0.9212	0.8622
DTs	0.9653	0.8453	0.9795	0.8486	0.9124	0.8464
GNB	0.9419	0.7811	0.9657	0.7738	0.8734	0.7667

precisely, it achieved an average F-Score improvement of 19% on the GNB classifier and 11% on the Decision Trees. In addition, the RF and SVM classifiers, which have obtained good classification results, have been outperformed by the federated detector with an average improvement of 5% and 9%, respectively. However, differently from our approach, such methods have been trained on the centralized raw data arranged as adjacent matrices. Therefore, they need an extra pre-processing step as well as not guarantee the privacy of the involved clients and the confidentiality of related data.

In addition, the adjacent matrices (73×73) have been used to test also a DL-based approach. We accomplished this through a Keras API-based CNN, shown in Fig. 8, trained respectively in centralized and federated-learning scenarios. Table 10 summarizes the results derived by considering each splitting criteria and the number of subscribed clients, Table 11 reports the client-related

Table 10
Performance results related to CNN (only Avg. values are reported).

Split	Acc.	Sens.	Spec.	Prec.	AUC	F-Mea.
Centr.	0.9697	0.8692	0.9820	0.8741	0.9256	0.8711
Mix4	0.9695	0.8705	0.9819	0.8754	0.9262	0.8714
Mix8	0.9659	0.8585	0.9799	0.8576	0.9192	0.8563
Mix12	0.9635	0.8467	0.9784	0.8486	0.9126	0.8462
Mix16	0.9603	0.8318	0.9766	0.8325	0.9042	0.8297

Table 11
Clients-related accuracy values.

Clients	Split	min_Acc.	max_Acc.	avg_Acc.
4	Mix	0.9678	0.9687	0.9683
8	Mix	0.9638	0.9655	0.9647
12	Mix	0.9595	0.9622	0.9613
16	Mix	0.9561	0.9595	0.9583

Table 12
Comparison between our approach and the CNN (only best values are reported).

Method	Acc.	Sens.	Spec.	Prec.	AUC	F-Mea.
Proposed	0.9920	0.9510	0.9952	0.9591	0.9731	0.9549
Centr.	0.9697	0.8692	0.9820	0.8741	0.9256	0.8711
Mix4	0.9695	0.8705	0.9819	0.8754	0.9262	0.8714

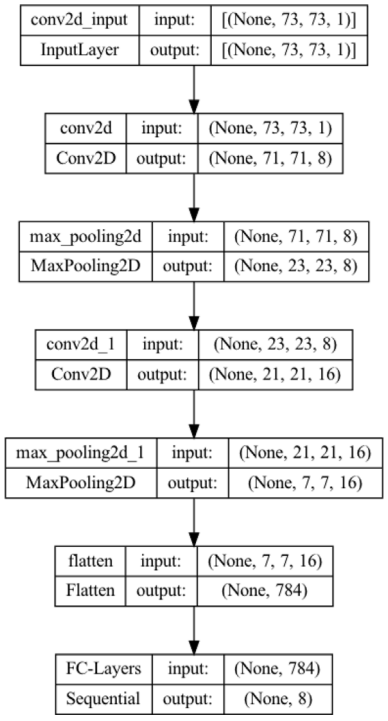

Fig. 8. Architecture of the CNN used in comparisons.

Accuracy values derived on the testing dataset, while Table 12 compares our approach with the above CNN.

As shown in Tables 10 and 11, the CNN achieved comparable results only with the Mixed splitting criteria (Mix). Vice-versa, due to the lack of convergence of the Federated CNN, no relevant results have been achieved by the other splitting criteria (Vertical and Horizontal). More precisely, the Vertical one has highlighted how DL-based models cannot learn if the data-related categories are distributed only to some clients, adversely affecting the federated learning process. Instead, the Horizontal one has highlighted how IID data can also adversely influence the FL-based models. This phenomenon occurs when data are characterized from many

Table 13
Horizontal training set division for each client.

Clients	Air	DKF	Fus	Gen	Gin	Jis	Opf	Sms
4	66	175	29	55	93	94	107	30
8	33	87	14	27	46	47	53	15
12	22	58	9	18	31	31	35	10
16	16	43	7	13	23	23	26	7

Table 14
Vertical training set division for 4 clients.

Client	Air	DKF	Fus	Gen	Gin	Jis	Opf	Sms
C1	265	700	0	0	0	0	0	0
C2	0	0	117	220	0	0	0	0
C3	0	0	0	0	372	376	0	0
C4	0	0	0	0	0	0	431	122

Table 15
Mixed training set division for 4 clients.

Client	Air	DKF	Fus	Gen	Gin	Jis	Opf	Sms
C1	265	350	0	0	0	0	107	0
C2	0	350	117	0	0	0	107	0
C3	0	0	0	110	186	188	107	0
C4	0	0	0	110	186	188	110	122

Table 16
Vertical training set division for 8 clients.

Client	Air	DKF	Fus	Gen	Gin	Jis	Opf	Sms
C1	132	350	0	0	0	0	0	0
C2	0	0	58	110	0	0	0	0
C3	0	0	0	0	186	188	0	0
C4	0	0	0	0	0	0	215	61
C5	133	350	0	0	0	0	0	0
C6	0	0	59	110	0	0	0	0
C7	0	0	0	0	186	188	0	0
C8	0	0	0	0	0	0	216	61

sub-categories, producing a sort of nested Vertical splitting. In our case, this means that each k-spaced adjacent matrix can be related to a specific k-th sub-category, with k varying from 1 to 100. Therefore, these matrices adversely affect the federated learning process also when they seem to be Independent and Identically Distributed (IID). Finally, the comparison provided in Table 12 shows that our approach outperformed the FL-based CNN by achieving an average F-Score improvement of 8%. Therefore, this confirms the effectiveness of the proposed detector in achieving excellent classification metrics for any considered splitting criteria (IID and no-IID) that, instead, adversely affect the classical FL-based solutions [11–14].

5.5. Performance evaluation

To evaluate the performance of the proposed architecture during the Client-Side Extraction and Server-Side Aggregation processes, we first derived the required time effort by using the entire training set on a single machine. Next, we partitioned it by applying the previously mentioned splitting criteria and considering the number of involved clients, as reported in Tables 13–21, respectively.

Hence, to carefully analyze the model scalability for each considered combination (splitting criteria/number of clients), we derived the parallel Speedup [41], which is given by:

$$Speedup = \frac{T_s}{T_p}, \tag{17}$$

Table 17
Mixed training set division for 8 clients.

Client	Air	DKF	Fus	Gen	Gin	Jis	Opf	Sms
C1	132	175	0	0	0	0	107	0
C2	0	175	117	0	0	0	107	0
C3	0	0	0	55	93	94	107	0
C4	0	0	0	55	93	94	110	122
C5	133	175	0	0	0	0	0	0
C6	0	175	0	0	0	0	0	0
C7	0	0	0	55	93	94	0	0
C8	0	0	0	55	93	94	0	0

Table 18
Vertical training set division for 12 clients.

Client	Air	DKF	Fus	Gen	Gin	Jis	Opf	Sms
C1	66	175	0	0	0	0	0	0
C2	0	0	29	55	0	0	0	0
C3	0	0	0	0	93	94	0	0
C4	0	0	0	0	0	0	215	61
C5	133	350	0	0	0	0	0	0
C6	0	0	59	110	0	0	0	0
C7	0	0	0	0	186	188	0	0
C8	0	0	0	0	0	0	108	30
C9	66	175	0	0	0	0	0	0
C10	0	0	29	55	0	0	0	0
C11	0	0	0	0	93	94	0	0
C12	0	0	0	0	0	0	108	31

Table 19
Mixed training set division for 12 clients.

Client	Air	DKF	Fus	Gen	Gin	Jis	Opf	Sms
C1	66	175	0	0	0	0	53	0
C2	0	175	29	0	0	0	53	0
C3	0	0	0	55	93	94	53	0
C4	0	0	0	55	93	94	53	30
C5	133	175	0	0	0	0	0	0
C6	0	175	0	0	0	0	0	0
C7	0	0	0	55	93	94	0	0
C8	0	0	0	55	93	94	0	0
C9	66	0	0	0	0	0	53	0
C10	0	0	29	0	0	0	53	30
C11	0	0	29	0	0	0	53	30
C12	0	0	30	0	0	0	60	32

Table 20
Vertical training set division for 16 clients.

	Air	DKF	Fus	Gen	Gin	Jis	Opf	Sms
C1	66	175	0	0	0	0	0	0
C2	0	0	29	55	0	0	0	0
C3	0	0	0	0	93	94	0	0
C4	0	0	0	0	0	0	107	30
C5	66	175	0	0	0	0	0	0
C6	0	0	29	55	0	0	0	0
C7	0	0	0	0	93	94	0	0
C8	0	0	0	0	0	0	107	30
C9	66	175	0	0	0	0	0	0
C10	0	0	29	55	0	0	0	0
C11	0	0	0	0	93	94	0	0
C12	0	0	0	0	0	0	107	30
C13	67	175	0	0	0	0	0	0
C14	0	0	30	55	0	0	0	0
C15	0	0	0	0	93	94	0	0
C16	0	0	0	0	0	0	110	32

where T_s is the time effort without parallelism and T_p is the required time effort with parallelism. Fig. 9 shows the speedup-related behavior for each considered splitting criteria and clients, respectively.

As shown in Fig. 9, since the derived Speedup values are slightly different, it is possible to appreciate how the proposed model is characterized by semi-linear scalability. More precisely,

Table 21
Mixed training set division for 16 clients.

	Air	DKF	Fus	Gen	Gin	Jis	Opf	Sms
C1	66	87	0	0	0	0	53	0
C2	0	87	29	0	0	0	53	0
C3	0	0	0	55	93	94	53	0
C4	0	0	0	55	93	94	53	30
C5	66	87	0	0	0	0	0	0
C6	0	87	0	0	0	0	0	0
C7	0	0	0	55	93	94	0	0
C8	0	0	0	55	93	94	0	0
C9	66	0	0	0	0	0	53	0
C10	0	0	29	0	0	0	53	30
C11	0	0	29	0	0	0	53	30
C12	0	0	30	0	0	0	60	32
C13	67	87	0	0	0	0	0	0
C14	0	87	0	0	0	0	0	0
C15	0	87	0	0	0	0	0	0
C16	0	91	0	0	0	0	0	0

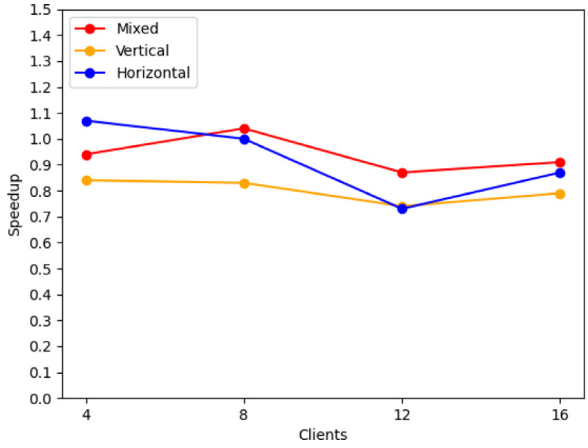

Fig. 9. Speedup comparison.

when the number of clients is low (i.e. 4 and 8), the system rapidly scales thanks to low communication costs. Instead, when the number of clients increases, the related execution costs decrease to the point that counterbalances the high communication costs. For this reason, differently from the classical FL-based solutions [11–14], the proposed architecture can obtain excellent time performances in the presence of non-IID.

6. Conclusions and future work

In this paper, we investigated the capabilities of Markov chains and associations-based rules within an IoT environment in order to support a privacy-aware malware classification. To accomplish this, we enhanced the dynamic-based detector, presented in [3], according to a federated logic. Therefore, we proposed a dedicated architecture capable of performing a federated training process in which end-users build a shared detector by sending the analyzed applications to a central server. Consequently, the interchange of sensitive information, and their protection against the most common data leakage threats, has been similarly guaranteed as done in traditional Federated Learning-based approaches. Next, we validated the effectiveness of the presented method on several famous malware families derived from the UMD dataset. More precisely, the obtained results have shown an average accuracy of 99% by outperforming the most famous DL and FL-based approaches and highlighting possible benefits in zero-day malware detection. Finally, we also provided a statistical and temporal performance assessment in the presence of non-IID data, in which several dataset partitions, splitting criteria, and the number of subscribed clients have been considered, respectively.

The main limitation of the proposed approach is related to the role of the server as a knowledge aggregator that can become a single point of failure. This can also result in a significant load in presence of a growing number of clients whose locally-inferred models significantly differ. Also in the lack of a common synchronization mechanism between the server and all its clients, the server could experience a continuous update of the rule-based model, resulting in unnecessary overhead.

However, due to the high number of existing and yearly released malware applications, this study proposes some possible future works that might provide some benefits, such as well-suited mechanisms to improve the zero-day detection and the building of new communication channels and environments for federated entities, respectively.

For this reason, we will investigate an extension of the proposed detector to detect new malware families and variants characterized by more intricate dynamic patterns. For instance, given an arbitrary rule, the pruning indexes might consider only the "nearest rules". Also, we will optimize the applications-related extraction process in order to improve the required temporal effort. For instance, several selection criteria could be used (e.g. the proposed pruning indexes) to consider only the relevant rule sequences effectively mined during the application execution.

CRediT authorship contribution statement

Gianni D'Angelo: Supervision, Methodology, Experimentals, Software supervision, Writing and reviewing. **Eslam Farsimadan:** Investigation, Methodology, Writing – original draft. **Massimo Ficco:** Investigation, Conceptualization, Methodology, Reviewing. **Francesco Palmieri:** Supervision, Conceptualization, Methodology, Writing and reviewing. **Antonio Robustelli:** Investigation, Conceptualization, Writing – original draft, Experimentals, Software.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This work was partially supported by the GERONIMO Project within the Research and Innovation Program PE00000014, CUP D43C22003050001 "SEcurity and RIghts in the CyberSpace (SERICS)". The program is funded by the European Union – Next Generation EU, within the "Extended Partnership" PNRR initiative. All authors approved the version of the manuscript to be published.

References

[1] L. Nataraj, S. Karthikeyan, B. Manjunath, Sattva: Sparsity inspired classification of malware variants, in: Proceedings of the 3rd ACM Workshop on Information Hiding and Multimedia Security, IH&MMSec '15, Association for Computing Machinery, New York, NY, USA, 2015, pp. 135–140, http://dx.doi.org/10.1145/2756601.2756616.

[2] G. D'Angelo, M. Ficco, F. Palmieri, Malware detection in mobile environments based on autoencoders and api-images, J. Parallel Distrib. Comput. 137 (2020) 26–33, http://dx.doi.org/10.1016/j.jpdc.2019.11.001.

[3] G. D'Angelo, M. Ficco, F. Palmieri, Association rule-based malware classification using common subsequences of api calls, Appl. Soft Comput. 105 (2021) 107234, http://dx.doi.org/10.1016/j.asoc.2021.107234.

[4] H.-Y. Chuang, S.-D. Wang, Machine learning based hybrid behavior models for android malware analysis, in: 2015 IEEE International Conference on Software Quality, Reliability and Security, IEEE, 2015, pp. 201–206.

[5] N. Zhang, K. Yuan, M. Naveed, X. yong Zhou, X. Wang, Leave me alone: App-level protection against runtime information gathering on android, in: IEEE Symposium on Security and Privacy, IEEE Computer Society, 2015, pp. 915–930.

[6] M. Ficco, Detecting iot malware by markov chain behavioral models, in: 2019 IEEE International Conference on Cloud Engineering (IC2E), 2019, pp. 229–234, http://dx.doi.org/10.1109/IC2E.2019.00037.

[7] A. Martín, V. Rodríguez-Fernández, D. Camacho, Candyman: Classifying android malware families by modelling dynamic traces with markov chains, Eng. Appl. Artif. Intell. 74 (2018) 121–133, http://dx.doi.org/10.1016/j.engappai.2018.06.006, URL https://www.sciencedirect.com/science/article/pii/S0952197618301374.

[8] R. Galvez, V. Moonsamy, C. Díaz, Less is more: A privacy-respecting android malware classifier using federated learning, CoRR abs/2007.08319. arXiv:2007.08319. URL https://arxiv.org/abs/2007.08319.

[9] R.-H. Hsu, Y.-C. Wang, C.-I. Fan, B. Sun, T. Ban, T. Takahashi, T.-W. Wu, S.-W. Kao, A privacy-preserving federated learning system for android malware detection based on edge computing, in: 2020 15th Asia Joint Conference on Information Security (AsiaJCIS), 2020, pp. 128–136, http://dx.doi.org/10.1109/AsiaJCIS50894.2020.00031.

[10] V. Rey, P.M. Sánchez Sánchez, A. Huertas Celdrán, G. Bovet, Federated learning for malware detection in iot devices, Comput. Netw. 204 (2022) 108693, http://dx.doi.org/10.1016/j.comnet.2021.108693.

[11] S.P. Karimireddy, S. Kale, M. Mohri, S. Reddi, S. Stich, A.T. Suresh, SCAFFOLD: Stochastic controlled averaging for federated learning, in: H.D. III, A. Singh (Eds.), Proceedings of the 37th International Conference on Machine Learning, in: Proceedings of Machine Learning Research, vol. 119, PMLR, 2020, pp. 5132–5143.

[12] T. Li, A.K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, V. Smith, Federated optimization in heterogeneous networks, in: Proceedings of Machine Learning and Systems, Vol. 2, 2020, pp. 429–450.

[13] R. Pathak, M.J. Wainwright, Fedsplit: an algorithmic framework for fast federated optimization, in: H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, H. Lin (Eds.), Advances in Neural Information Processing Systems, Vol. 33, Curran Associates, Inc., 2020, pp. 7057–7066.

[14] Z. Charles, J. Konečný, On the outsized importance of learning rates in local update methods, 2020, http://dx.doi.org/10.48550/ARXIV.2007.00878, URL https://arxiv.org/abs/2007.00878.

[15] K. Tam, A. Fattori, S. Khan, L. Cavallaro, Copperdroid: Automatic reconstruction of android malware behaviors, in: NDSS Symposium 2015, NDSS Symposium 2015, 2015, pp. 1–15, http://dx.doi.org/10.14722/ndss.2015.23145.

[16] O.E. David, N.S. Netanyahu, Deepsign: Deep learning for automatic malware signature generation and classification, in: 2015 International Joint Conference on Neural Networks, IJCNN, 2015, pp. 1–8, http://dx.doi.org/10.1109/IJCNN.2015.7280815.

[17] G. D'Angelo, F. Palmieri, A. Robustelli, Effectiveness of video-classification in android malware detection through api-streams and cnn-lstm autoencoders, in: I. You, H. Kim, T.-Y. Youn, F. Palmieri, I. Kotenko (Eds.), Mobile Internet Security, Springer Singapore, Singapore, 2022, pp. 171–194.

[18] A. Elhadi, M. Maarof, B. Barry, Improving the detection of malware behaviour using simplified data dependent api call graph, Int. J. Secur. Appl. 7 (2013) 29–42, http://dx.doi.org/10.14257/ijsia.2013.7.5.03.

[19] Y. Ding, X. Xia, S. Chen, Y. Li, A malware detection method based on family behavior graph, Comput. Secur. 73 (2018) 73–86, http://dx.doi.org/10.1016/j.cose.2017.10.007.

[20] B. McMahan, E. Moore, D. Ramage, S. Hampson, B.A. y. Arcas, Communication-efficient learning of deep networks from decentralized data, in: A. Singh, J. Zhu (Eds.), Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, in: Proceedings of Machine Learning Research, vol. 54, PMLR, 2017, pp. 1273–1282.

[21] Q. Yang, Y. Liu, T. Chen, Y. Tong, Federated machine learning: Concept and applications, ACM Trans. Intell. Syst. Technol. 10 (2). http://dx.doi.org/10.1145/3298981.

[22] A. Cholakoska, B. Pfitzner, H. Gjoreski, V. Rakovic, B. Arnrich, M. Kalendar, Differentially private federated learningfor anomaly detection in ehealth networks, in: Adjunct Proceedings of the 2021 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2021 ACM International Symposium on Wearable Computers, Association for Computing Machinery, New York, NY, USA, 2021, pp. 514–518.

[23] M. Dhada, A.K. Jain, A.K. Parlikad, Empirical convergence analysis of federated averaging for failure prognosis, IFAC-PapersOnLine 53 (3) (2020) 360–365, http://dx.doi.org/10.1016/j.ifacol.2020.11.058, (this research was funded by the epsrc and bt prosperity partnership project: Next generation converged digital infrastructure, grant number ep/r004935/1), 4th IFAC Workshop on Advanced Maintenance Engineering, Services and Technologies - AMEST 2020.

[24] Y. Zhao, J. Chen, D. Wu, J. Teng, S. Yu, Multi-task network anomaly detection using federated learning, in: Proceedings of the Tenth International Symposium on Information and Communication Technology, SoICT 2019, Association for Computing Machinery, New York, NY, USA, 2019, pp. 273–279, http://dx.doi.org/10.1145/3368926.3369705.

[25] T. Zhang, C. He, T. Ma, L. Gao, M. Ma, S. Avestimehr, Federated learning for internet of things, in: Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems, SenSys '21, Association for Computing Machinery, New York, NY, USA, 2021, pp. 413–419, http://dx.doi.org/10.1145/3485730.3493444.

[26] L.U. Khan, W. Saad, Z. Han, E. Hossain, C.S. Hong, Federated learning for internet of things: Recent advances, taxonomy, and open challenges, IEEE Commun. Surv. Tutor. 23 (3) (2021) 1759–1799, http://dx.doi.org/10.1109/COMST.2021.3090430.

[27] D.C. Nguyen, M. Ding, P.N. Pathirana, A. Seneviratne, J. Li, H. Vincent Poor, Federated learning for internet of things: A comprehensive survey, IEEE Commun. Surv. Tutor. 23 (3) (2021) 1622–1658, http://dx.doi.org/10.1109/COMST.2021.3075439.

[28] S. Shukla, P.D. Sai Manoj, G. Kolhe, S. Rafatirad, On-device malware detection using performance-aware and robust collaborative learning, in: 2021 58th ACM/IEEE Design Automation Conference, DAC, 2021, pp. 967–972, http://dx.doi.org/10.1109/DAC18074.2021.9586330.

[29] S.I. Popoola, R. Ande, B. Adebisi, G. Gui, M. Hammoudeh, O. Jogunola, Federated deep learning for zero-day botnet attack detection in iot-edge devices, IEEE Internet Things J. 9 (5) (2022) 3930–3944, http://dx.doi.org/10.1109/JIOT.2021.3100755.

[30] Y. Lu, Z. Liu, Y. Huang, Parameters compressed mechanism in federated learning for edge computing, in: 2021 8th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2021 7th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom), 2021, pp. 161–166, http://dx.doi.org/10.1109/CSCloud-EdgeCom52276.2021.00038.

[31] G. Paragliola, A. Coronato, Definition of a novel federated learning approach to reduce communication costs, Expert Syst. Appl. 189 (2022) 116109, http://dx.doi.org/10.1016/j.eswa.2021.116109.

[32] G.A. Fink, Markov Models for Pattern Recognition, second ed., in: Advances in Computer Vision and Pattern Recognition, Springer, London, England, 2014.

[33] B. McMahan, E. Moore, D. Ramage, S. Hampson, B.A. y Arcas, Communication-efficient learning of deep networks from decentralized data, in: Artificial Intelligence and Statistics, 2017, pp. 1273–1282.

[34] H. Wang, M. Yurochkin, Y. Sun, D. Papailiopoulos, Y. Khazaeni, Federated learning with matched averaging, 2020, http://dx.doi.org/10.48550/ARXIV.2002.06440, URL https://arxiv.org/abs/2002.06440.

[35] S. Ek, F. Portet, P. Lalanda, G. Vega, A federated learning aggregation algorithm for pervasive computing: Evaluation and comparison, in: 2021 IEEE International Conference on Pervasive Computing and Communications (PerCom), IEEE, 2021, pp. 1–10, http://dx.doi.org/10.1109/percom50583.2021.9439129.

[36] F. Carrez, T. Elsaleh, D. Gómez, L. Sánchez, J. Lanza, P. Grace, A reference architecture for federating iot infrastructures supporting semantic interoperability, in: 2017 European Conference on Networks and Communications (EuCNC), 2017, pp. 1–6, http://dx.doi.org/10.1109/EuCNC.2017.7980765.

[37] C.-S. Shih, C.-C. Chuang, H.-Y. Yeh, Federating public and private intelligent services for iot applications, in: 2017 13th International Wireless Communications and Mobile Computing Conference, IWCMC, 2017, pp. 558–563, http://dx.doi.org/10.1109/IWCMC.2017.7986346.

[38] M.N. Sadat, M.M. Al Aziz, N. Mohammed, F. Chen, X. Jiang, S. Wang, Safety: Secure gwas in federated environment through a hybrid solution, IEEE/ACM Trans. Comput. Biol. Bioinform. 16 (1) (2019) 93–102, http://dx.doi.org/10.1109/TCBB.2018.2829760.

[39] G. D'Angelo, F. Palmieri, A. Robustelli, A. Castiglione, Effective classification of android malware families through dynamic features and neural networks, Connect. Sci. 33 (3) (2021) 786–801, http://dx.doi.org/10.1080/09540091.2021.1889977, arXiv:https://doi.org/10.1080/09540091.2021.1889977.

[40] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, J. Mach. Learn. Res. 12 (2011) 2825–2830.

[41] Selkie, Parallel speedup, 2021, URL http://selkie.macalester.edu/csinparallel/modules/IntermediateIntroduction/build/html/ParallelSpeedup/ParallelSpeedup.html.

Gianni D'Angelo received a Ph.D. degree in Computer Science, applied electromagnetism and telecommunications from the University of Salerno, Italy, where he is currently a Tenured Adjunct Professor (RTD-B). His research interests concern the development and implementation of Soft Computing algorithms for High-Performance machines, HPC, and Parallel Computing for Knowledge Discovery in the big data context. He gained experience in the world of pattern recognition, neural networks, fuzzy logic, ANFIS systems, genetic and evolutionary algorithms, and parallel programming applied in various scientific and industrial fields. He authored many articles published in international journals, books and conferences, as well as currently serves as a reviewer, editorial board and guest editor for several international journals.

Eslam Farsimadan received an M.S. degree in soft computing from Shahid Beheshti University, Tehran, Iran, and his first Ph. D degree in mathematical logic from Shahid Bahonar University, Kerman, Iran. He is currently pursuing his second Ph. D in computer science at the University of Salerno, Italy. His research interests concern high-performance networking protocols, network security, and control theory and optimization. He serves as the reviewer of several international journals. He is a member of the Iranian Mathematical Society (IMS) and IEEE.

Massimo Ficco is a Full Professor at the University of Salerno, Italy. He received a Ph.D. degree in computer engineering from the University of Naples "Parthenope", Italy. Previously he was 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 and board-editor 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.

Francesco Palmieri is a full professor at the University of Salerno, Italy. He received from the same university a Ph.D. in computer science. His major research interests concern high performance networking protocols and architectures, routing algorithms and network security. Previously he has been an assistant professor at the Second University of Naples, and the Director of the telecommunication and networking division of the Federico II University, in Naples, Italy. At the start of his career, he also worked for several international companies on networking-related projects. He has been closely involved with the development of the Internet in Italy as a senior member of the Technical–Scientific Advisory Committee and of the CSIRT of the Italian NREN GARR. He has published a large number of papers (more than 250) in leading technical journals, books and conferences and currently serves as the editor-in-chief of an international journal and is part of the editorial board or associate editor of several other well reputed ones.

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