A Knowledge Transfer-Based Semi-Supervised Federated Learning for IoT Malware Detection

Xinjun Pei, Xiaoheng Deng, *Member, IEEE*, Shengwei Tian, Lan Zhang, *Member, IEEE*, and Kaiping Xue, *Senior Member, IEEE*

Abstract—As the demand for Internet of Things (IoT) technologies continues to grow, IoT devices have been viable targets for malware infections. Although deep learning-based malware detection has achieved great success, the detection models are usually trained based on the collected user records, thereby leading to significant privacy risks. One promising solution is to leverage federated learning (FL) to enable distributed on-device training without centralizing the private user records. However, it is non-trivial for IoT users to label these records, where the quality and the trustworthiness of data labeling are hard to guarantee. To address the above issues, this paper develops a semi-supervised federated IoT malware detection framework based on knowledge transfer technologies, named by FedMalDE. Specifically, FedMalDE explores the underlying correlation between labeled and unlabeled records to infer labels towards unlabeled samples by the knowledge transfer mechanism. Moreover, a specially designed subgraph aggregated capsule network (SACN) is used to efficiently capture varied malicious behaviors. The extensive experiments conducted on real-world data demonstrate the effectiveness of FedMalDE in detecting IoT malware and its sufficient privacy and robustness guarantee.

Index Terms—Malware detection, federated learning, semi-supervised learning, privacy-preserving, capsule network

1 Introduction

Recent years have witnessed the evolution of Internet of Things (IoT) technologies. Typically, a vast amount of IoT devices are managed and controlled by a central server to share information and improve IoT user experiences, which, however, opens up the possibility of new attacks targeting IoT devices. Due to the lack of basic security monitoring and protection mechanisms [1], malware attacks have become very common in IoT environments, leading to serious privacy leakage, device hijacking, unauthorized access, and many other security issues.

To mitigate malware threats, deep learning has become one indispensable and well-recognized solution, which leverages a large number of records collected from users to obtain useful patterns and trends of malware attacks in a centralized manner [1], [2]. In the centralized setting, apps are often uploaded to the cloud server for detection before the installation, which brings a substantial network overhead and costs (especially for large apps), and also involves insecure private data collection and uploading. One promising solution is to enable distributed learning to leverage users' sensitive data locally without migrating the private IoT mobile data to the cloud server, which is the so called Federated Learning (FL) [3], [4], [5].

Nevertheless, most FL approaches assume the availability of a large amount of labeled data [6], which becomes impractical for IoT malware detection. Specifically, labeling malware data is expensive due to the required high-level of human expertise that few IoT users have. Moreover, malicious users may inject mobile records with poisoned labels to contaminate the detection model [5], [7], [8]. To address the above concerns, we propose a semi-supervised FL framework to enable automatic labeling for privacy-preserving IoT malware detection.

A few recent efforts have been put on semi-supervised FL [9], [10]. For instance, Jeong *et al.*, [9] developed a general framework (FedMatch) by enforcing the consistency between predictions made across multiple user models for semi-supervised FL. The most relevant work to us is LiM [10], which establishes a decentralized malware classifier by learning from data without any domain knowledge guidance. However, the generated noisy pseudo-labels can mislead the detection model to forget the knowledge represented by the ground truth labeled data. To alleviate such issues in semi-supervised learning, knowledge transfer technologies [11], [12] have attracted recent attention to increase the accuracy of the generated pseudo-labels and thus improve the robustness of the learned models. Inspired

- *Xinjun Pei and Xiaoheng Deng are with the School of Computer Science and Engineering and Shenzhen Research Institute, Central South University, Changsha 410083, China. E-mail: pei_xinjun@163.com, dxh@csu.edu.cn.*
- *Shengwei Tian is with the School of Software, Xinjiang University, Urumqi 830001, China. E-mail: tianshengwei@163.com.*
- *Lan Zhang is with the Department of Electrical and Computer Engineering, Michigan Technological University, Houghton, MI 49931 USA. E-mail: lanzhang@mtu.edu.*
- *Kaiping Xue is with the Department of Electronic Engineering and Information Science, University of Science and Technology of China, Hefei 230027, China. E-mail: kpxue@ustc.edu.cn.*

Manuscript received 26 Nov. 2021; revised 25 Mar. 2022; accepted 4 May 2022. Date of publication 10 May 2022; date of current version 13 May 2023.
This work was supported in part by the National Natural Science Foundation of China under Grants 62172441, 62172449, and 61772553, in part by Local Science and Technology Developing Fundation Guided by Central Goverment through Free Exploration Project under Grant 2021Szvup166, in part by Autonomous Region Key R&D Project under Grant 2021B01002, and in part by the Fundamental Research Funds for the Central Universities of Central South University under Grant 2021zzts0201.
(Corresponding author: Xiaoheng Deng.)
Digital Object Identifier no. 10.1109/TDSC.2022.3173664

by this, we extend the conventional centralized knowledge transfer-based approach to federated settings by sharing the class probability information to enable robust detection based on semi-supervised learning.

Upon the proposed semi-supervised FL framework above, it is critical to develop an effective malware detection model. One feasible solution is to formulate the complicated malware detection as a graph learning problem. Most existing detection systems [13], [14] use a variety of graph mining-based techniques to depict higher-level semantic representations, such as function call graph (FCG) [15] and control flow graph (CFG) [16], [17]. These techniques mainly focus on method-level program semantics to extract fine-grained local features, which may cause scalability issues due to a large number of nodes in a given graph. Although they reduce the large dimension of a call graph into a low-dimensional feature vector, it is still difficult or unrealistic to reflect the entire graphic information. Instead, we intend to exploit higher-level program semantics by progressively aggregating fine-grained local features from a global perspective to obtain high-quality graph representations, where the subgraph-based representation method is used to model the program semantic and graph topology efficiently.

Based on the above ideas, this paper proposes a knowledge transfer-based semi-supervised federated learning framework for privacy-preserving IoT malware detection, named by FedMalDE. In many realistic scenarios, traditional analytic solutions like cloud-based malware detection are often used to store and process users' private records in a centralized manner, which raises serious privacy concerns. To overcome this challenge, this paper implements distributed FL to exploit user private data locally, avoiding the significant communication costs and the insecure data transmission in the conventional centralized learning. Moreover, using FL to proceed with malware detection faces a more realistic problem: it requires a larger amount of labeled data for practical collaborative training, which is a strong assumption that is hard to satisfy in practice. To relieve the need for data labeling in the federated system, we combine semi-supervised learning and federated learning to decentralize a malware classifier. Based on the knowledge transfer (teacher-student) approach, unlabeled user records can be automatically labeled, in which the cloud model (teacher) with prior malware knowledge can teach the distributed user models (students) to derive a reliable malware detection model in the federated setting. Besides, in order to learn a high-quality graph representation, we propose a subgraph aggregated capsule network (SACN) to capture various granularity graph information. To the best of our knowledge, this work is the first attempt to investigate the knowledge transfer-based semi-supervised IoT malware detection under the well-recognized federated setting. The main contributions of this paper are summarized as follows:

- We propose a semi-supervised federated IoT malware detection framework (FedMalDE) to exploit highly decentralized private data to collectively train a reliable detection model without revealing users' specific undisclosed information, which overcomes the limitation that the data uploading suffers from the security threats of attackers in the conventional centralized learning.
- We build a knowledge transfer mechanism to address the need for data labeling in the federated system, in which the teacher model with prior malware domain knowledge shares the class probability information to participating model trainers, and guides them to update the local student models during the federated training process.
- We develop a call subgraph-based representation method to model the program semantic and graph topology, and design a subgraph aggregated capsule network (SACN) to progressively aggregate the fine-grained subgraph features to learn high-quality graph representations.
- We conduct extensive experiments on real-world datasets to demonstrate the effectiveness of FedMalDE in detecting malware, which can even achieve comparable performance to the centralized detection model.

In the rest of the paper. In the rest of the paper, we overview the related work in Section 2. Then, we give some preliminaries in Section 3 and describe the system model and threat model in Section 4. Sections 5 and 6 give the semi-supervised FL security system in details. In Section 7, we evaluate the proposed system and analyze the experimental results. Finally, Section 8 concludes the paper.

2 Related Work

Graph-Based Malware Detection. The emergence of novel IoT malware brings high risks to users. Many recent studies [13], [14] have provided graph analysis methods for malware detection, such as control flow graph (CFG) [17], function call graph (FCG) [15], [18], [22], frequent sub-graph [15], opcode graph [19], PSI-graph [18], etc. These methods investigate software properties by inspecting manifest files and analyzing app source codes without running the apps. For instance, Cesare *et al.*, [17] decomposed a set of control flow graphs (CFGs) into fixed k-subgraphs to create feature vectors, and used a distance metric based on the minimum matching distance to construct a minimum sum weight matching between two sets of graphs. However, the detection performance of such a method strongly depends on the quality of the features. In order to avoid the complexity of CFG analysis, Nguyen *et al.*, [18] first extracted high-level features from function call graphs (FCGs), called PSI-Graph, by using a graph embedding technique (graph2vec), and then used machine learning methods to identify threat patterns. Similarly, Fun *et al.*, [15] constructed frequent subgraphs based on the FCGs at the fine-grained method level to present the local call relations among the functions (i.e., the caller-callee relationships). Their solution assigns weights based on the importance of API calls to discern malicious apps from benign ones. The technique is resilient against variants, even when the code is considerably confused. Moreover, Azmoodeh *et al.*, [19] proposed an OpCode graph-based deep learning approach for Internet of Battlefield Things (IoBT) malware detection. They first extracted OpCode sequence of IoBT malware and then transmuted OpCodes into a graph in which nodes are

TABLE 1
Comparison Between Malware Detection Methods

Method	Architecture	Learning strategy	Feature Extraction		Model	Classification ability	Applications
			Characterization	Complexity			
Cesare *et al.*, [17]	Centralized	Supervised	Control Flow Graph	High	DBM-Tree	Moderate	Malware Detection
Nguyen *et al.*, [18]	Centralized	Supervised	PSI-Graph &Function Call Graph	High	CNN	High	Malware Detection
Fun *et al.*, [15]	Centralized	Supervised	Frequent Subgraph & Function Call Graph	High	SVM	Moderate	Malware Detection
Azmoodeh *et al.*, [19]	Centralized	Supervised	Opcode Graph	High	CNN	High	Malware Detection
Hsu *et al.*, [20]	Decentralized	Supervised	Permissions & API Calls	Moderate	SVM	Low	Malware Detection
Nguyen *et al.*, [21]	Decentralized	Supervised	Packet Sequences	Low	GRU	Moderate	Network Intrusion Detection
Glvez *et al.*, [10]	Decentralized	Semi-Supervised	Permissions &Components	Low	LiM	Low	Malware Detection
Our work	Decentralized	Semi-Supervised	Sliced Subgraph	Moderate	SACN	High	Malware Detection

represented by OpCodes, and edges are generated based on the nodes' affinity in the disassembled file of each sample.

In summary, there exist three main barriers to apply the above graph-based techniques. First, although these techniques were effective in identifying real malware instances, they often do not consider node/function properties (e.g., the meaning of functions). Without this knowledge, it is difficult to understand more complex malicious behaviors. Second, most existing malware detection methods have focused on method-level local program semantics, while ignoring necessary class-level program semantics which is crucial for malware detection. Third, analyzing the entire CFG/FCG with thousands of nodes incurs a high computational overhead. To overcome these challenges, this paper develops a call subgraph-based SACN algorithm that exploits higher-level program semantics and structural information by progressively aggregating fine-grained local subgraphs to express more complex malicious behaviors. Table 1 lists some representative malware detection methods.

Federated Malware Detection. Most existing studies on malware detection depend on centralizing the training by uploading data to cloud [23], [24], which incurs serious privacy concerns for users [25] because the feature sets used by learning models implicitly reveal users' preferences which can be utilized to conduct targeted advertising. The shortcomings of centralized detection methods highlight an imperative need for distributed machine learning. To address these issues, federated learning provides an especially promising solution, which not only helps preserve privacy but also reduces network traffic congestion [26]. An early version of this concept was proposed by McMahan *et al.*, [27]. They proposed an iterative model averaging-based federated learning method that enables multiple users to train a shared model by aggregating locally-computed updates while keeping data in their mobile devices. However, there are relatively few studies on FL algorithms for security related tasks. Hsu *et al.*, [20] proposed a federated learning-based malware detection system that uses static analysis to extract permissions and API calls as features, and applies a support vector machine (SVM) to classify Android applications. Nguyen *et al.*, [21] employed a federated learning to anomaly-detection-based intrusion detection. Their solution is able to detect anomalous deviations in IoT devices communications. Considering a large number of unlabeled samples and the expensive cost of labeling big data, existing FL schemes still face enormous challenges in reality, requiring sufficient labeled samples for training. To relieve the need for data labeling in the federated system, this paper extends the traditional FL to the semi-supervised paradigm to enable automatic labeling for privacy-preserving IoT malware detection, benefiting from both private and public data.

Semi-Supervised Federated Learning. Although supervised learning is the most widely used learning mode based on the availability of labeled data samples, it is non-trivial to label a large amount of data, especially in the field of malware detection that requires human expertise and domain knowledge. To address the above challenges, semi-supervised learning takes advantage of both labeled and unlabeled data. Jeong *et al.*, [9] used the consistency loss and parameter decomposition methods to improve semi-supervised federated learning. Glvez *et al.*, [10] proposed a decentralized malware detection framework, which can learn from data without any domain knowledge. As a matter of fact, learning from the unlabeled data may lead to forgetting what the model learned from the labeled data. Moreover, Papernot *et al.*, [28] demonstrated a general semi-supervised learning strategy, the Private Aggregation of Teacher Ensembles (PATE) approach, which applies differential privacy to the training process of generative adversarial networks (GANs). In this strategy, a student model is trained using auxiliary, unlabeled non-sensitive data. Similarly, Zhang *et al.*, [29] extended the PATE approach, and developed a novel noisy label detection mechanism for semi-supervised model training to further improve student model performance when training with noisy labels. They modified the original PATE by adding an additional discriminator in the GANs. Typically, the current PATE-based methods use disjoint subsets of data to train an ensemble of

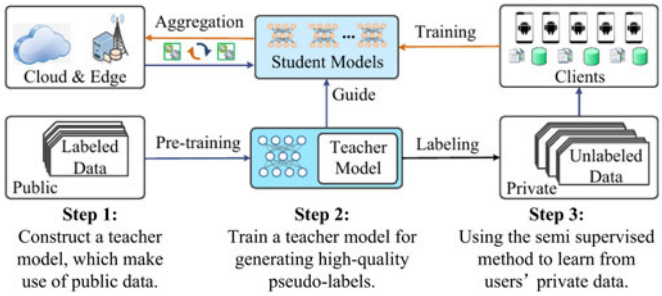


Fig. 1. The semi-supervised learning process.

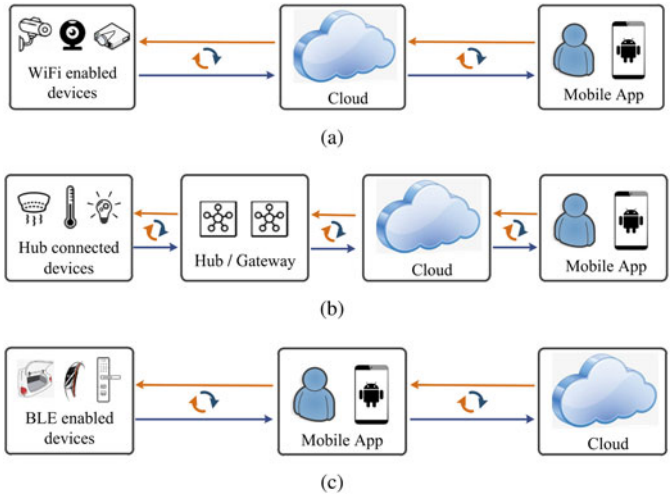


Fig. 2. Cloud-based IoT architectures.

teacher models, and then transfer the learned knowledge to students in a private manner. Specifically, they use the GAN model as a client-trained student model. However, the PATE's performance is mainly determined by the generator of GANs. The generator needs to capture the real distribution of instances, but unstable training makes this difficult to achieve. Besides, GANs may also suffer from the problem of mode collapse, i.e., the samples generated by the generator have similar properties, which makes the discriminator unable to distinguish between them, and thus severely affects the utility of PATE. In this paper, we propose a semi-supervised FL-based malware detection framework, which leverages a subgraph aggregated capsule network to improve detection performance. Moreover, we use a knowledge transfer mechanism to transfer the teacher's knowledge to the participating student models by sharing the class probability information, and thus improve the robustness of the learned models.

3 Preliminaries

In this section, we present the formal definition of federated learning and semi-supervised learning.

Federated Learning. Federated learning (FL) implements the distributed learning of the models at the decentralized client side, which aims to collaboratively train a global model in a privacy-preserved manner. Let G be a global model, and $S = \{s_1, s_2, \ldots, s_K\}$ be a set of local student models for K clients $Cli = \{cli_1, cli_2, \ldots, cli_K\}$. Let $D = \{x_i, y_i\}_{i=1}^{N}$ be a given dataset, where x_i is an arbitrary instance of application, N is the number of instances, and $y_i \in \{0, 1\}$ is the corresponding label of x_i, with $y_i = 0$ indicating a benign application, and $y_i = 1$ indicating a malware. In federated learning, clients collaboratively train the shared global model G using real-time generated mobile data without sharing their local data. After that, the cloud dynamically updates the global model in each training round by averaging the weights of all uploaded local models, i.e., $w \leftarrow \frac{n_{cli}}{n} \sum_{cli=1}^{Cli} w^{cli}$. Then, the global model minimizes its loss function $\mathcal{L}_G(w_{global})$. The learning procedure will be discussed later.

Semi-Supervised Learning. Generally, semi-supervised learning refers to learning from labeled training data and generalizing to available unlabeled (training) data. Fig. 1 depicts the semi-supervised learning process. The cloud server builds a teacher model to process and learn the data distribution and trends from labeled samples to obtain useful patterns. Then, the teacher model is sent to distributed IoT devices to generate high-quality pseudo-labels for clients' unlabeled private data. Afterwards, clients perform on-device learning using their private data that has been labeled with pseudo-labels, without centralizing the data. There is a dataset $D = \{x_i, y_i\}_{i=1}^{N}$, where x_i is the $i\text{-}th$ sample with the corresponding label y_i. We split this dataset into two sub-datasets D_T and D_S. Let $D_T = \{x_i, y_i\}_{i=1}^{N_t}$ be a set of labeled data instances, which is used to pre-train a teacher model at the cloud. Let $D_S = \{x_i\}_{i=1}^{N_s}$ be a set of unlabeled data instances (without the corresponding labels), which is used to train a set of local student models for K clients $Cli = \{cli_1, cli_2, \ldots, cli_K\}$. Specifically, in our case, D_S is composed of K sub-datasets $D_S^{cli_k} = \{x_i^{cli_k}\}_{i=1}^{N_s^{cli_k}}$ privately collected at each client cli_k. In a semi-supervised setting, each client does not contain the corresponding labels, and the labeled data can only be used at the cloud. Let $f_{w_{cli}}^{k}(x_i, y_i)$ be a neural network on the $k\text{-}th$ client cli_k, which is parameterized by weights w_{cli}, and gives the perdiction $\hat{y}_i$ for the input x_i. The goal is to minimize loss function $\mathcal{L}_S(w_{cli})$ on $D_S^{cli_k}$.

4 System Model

In this section, we first introduce the real world cloud-based IoT architectures followed by the considered threat model of malware attacks, based on which we present an idea of the proposed FedMalDE.

4.1 Cloud-Based IoT Architecture

Previous works [30], [31] have shown that IoT users have become accustomed to using mobile apps to interact with surrounding IoT devices. However, the importance of the role that mobile apps play in the IoT has always been underestimated. We argue that mobile apps do have an important impact on the interactions between IoT devices and their corresponding cloud. Its security needs to be thoroughly discussed. Therefore, we provide a holistic view of three cloud-based IoT architectures, and derive the IoT malware attack surfaces in the three architectures. Fig. 2 shows these IoT architectures.

Now we conduct a conceptual review of these IoT architectures. The cloud-based IoT architecture consists of IoT devices, an IoT cloud, and a mobile app. Briefly speaking,

the WiFi-enabled IoT devices first obtain Internet access to establish a connection with the cloud, and then negotiate login credentials with the cloud. Specifically, as shown in Figs. 2a and 2b, the IoT devices connected to the cloud can be divided into two types: i) the WiFi-enabled devices, which can be connected to the Internet and thus have the ability to communicate with the cloud instantaneously; ii) the energy-economic devices without WiFi interface, which use hubs/gateways to communicate with the cloud. These IoT devices regularly report their status and execute received remote control commands. After that, the cloud authorizes the users to remotely monitor and control each IoT device by using the device companion app in their smartphones. In this setting, when users send remote commands to IoT devices, the cloud with device control services can act as a "proxy" [31].

It is worth mentioning that IoT users have widely accepted mobile apps as interfaces to interact with surrounding IoT devices to issue commands and retrieve processed data, and even serve as gateways to provide Internet connectivity for a myriad of IoT devices in practice, e.g., smart home devices and wearable devices. As shown in Fig. 2c, the device companion app could interact with the IoT devices through the Bluetooth Low Energy (BLE) channel [32], while depending on the Internet connectivity to communicate with the cloud. In this case, the companion app acts as a "proxy" [30]. However, this interdependence may lead to a larger attack surface of malware.

4.2 Threat Model

Undoubtedly, the role of mobile apps complicates the IoT security design. More importantly, the fact that the app may manipulate the communication between the cloud and the IoT device greatly expands the malware attack surface. We list possible malware attacks against the above three IoT architectures, which pose a serious threat to the entire workflow of the IoT.

- *A1* - A malicious app could access external devices (e.g., Bluetooth device) by the channel shared with the device companion app due to the coarse-grained permissions provided by the Android system.
- *A2* - A malicious app that has been granted Bluetooth permissions could launch a co-location attack, where a co-located attacker can manipulate the traffic of a companion app that co-locates with the malicious app [32].
- *A3* - A malicious app could create some simple rules to instruct the cloud to automatically interact with the device, and triggers those rules at a specific time, for example, to turn on the smart lock at 7 am. In the worst case it can even be life-threatening, such as the operation of a cardiac descender.
- *A4* - A malicious app may request more permissions than needed, and execute implicit code paths, which leads to implicit trigger-action chains to be fired immediately.
- *A5* - A malicious app could blind the cloud, or make the device out of sync with the cloud by deliberately blocking certain messages.
- *A6* - A malicious app could tamper with the information that the cloud uses to authenticate the IoT device, which could cause serious consequences, such as sensitive information leakage, illegal access to the device, and data injection attack.
- *A7* - Device owner authorizes some temporary users to access their devices for a certain period of time. When the authorization expires, the temporary users will have access revoked. However, a malicious app could exploit the leaked sensitive information to enable temporary users to access the device stealthily, even if authorization is revoked [33].

We consider various types of adversaries who might launch attacks against vulnerable IoT devices in the IoT network by malware. As new/unknown malware attacks continue to emerge, IoT malware detection systems become vital to adopt. However, there are many challenges entailed with such practices.

Challenges. We list some of the challenges of deploying a complete and scalable IoT malware detection system in IoT scenarios.

- *C1* - Existing deep learning-based malware detection systems usually involve processing and learning a large, diverse set of examples from public data sets to obtain useful threat patterns. The growth rate and evolution of malware highlights an urgent need to update existing detection models. However, cloud data is usually sensitive and private. For deep learning practitioners that cannot access large clouds, the challenge of collecting enough data to train deep learning models seems to never end. In this case, any supervised learning method would be difficult to implement.
- *C2* -Considering that traditional centralized detection schemes have only one attack detector entity (e.g., a centralized cloud responsible for training and attack detection), it makes the training and retraining of the learning models very challenging, especially when handling the massive amount of data generated by geographically distributed IoT devices. The back-and-forth communication of data imposes other problems, for example, the latency for data transmission is often very large because of the excessive communication overhead. Therefore, decentralized detection solutions are very much in need.
- *C3* - As discussed earlier, many real-world applications do not have access to large training sets because of data scarcity, or because of the difficulty and expense in labeling data. However, in a federated setting, we cannot expect users to assign correct labels to samples, especially in malware detection which requires high-level human expertise and domain knowledge that few users possess [6]. How to derive a more robust malware detection model from the massive unlabeled data that exists in reality is still a difficult and wide-open problem.
- *C4* - New IoT applications are released on a daily basis, many of which are downloaded from third-party markets that have not undergone rigorous malware screening. Meantime, adversaries are always

developing new malware attacks with varying degrees of complexity to circumvent security products. This makes the security of such IoT applications highly unpredictable and uncontrollable. To be useful in practice, our system must be adaptable in kind to resist malware.

4.3 Overview of FedMalDE

Consider that the cloud provides various sub-services and components for many organizations and services. We deploy the FedMalDE framework on the cloud to provide security services for users, which consists of Security Cloud and IoT Security Service. i) The role of Security Cloud is to monitor devices and perform malware detection for identifying potential risks related to malware infection in the network. ii) The IoT Security Service supported by service providers like Google and Microsoft maintains a repository of malware detection models that are trained locally by all clients in the federated system. Specifically, the IoT Security Service aggregates all these local models to update a global detection model on the Security Cloud. Our main security objective is to detect malware on IoT devices and prevent targeted devices from being infected.

Design Choices. In this paper, we build a privacy-preserving federated learning framework for IoT malware detection to perform local training and inference of detection models, which can utilize all available data for learning. As a result, our framework enables users to collaboratively train a shared model using real-time generated on-device data. This design choice enables the analysis fast for such enormous data without migrating the private end-user data, which reduces the network traffic congestion, effectively addressing challenges C1 and C2. Moreover, as described in challenge C3, it is hard to obtain large-scale, high-quality labeled data for practical collaborative training, which makes the application of FL limited. To relieve the need for data labeling in FL systems (challenge C3), we introduce the concepts of semi-supervised learning for FL, leading to a more practical and competitive paradigm, which leverages very limited labeled records in the cloud and massive unlabeled mobile data distributed across user devices to derive a reliable detection model. Moreover, we also develop a call subgraph-based SACN algorithm as an integral part of FL to identify threat patterns by progressively aggregating the fine-grained program semantics, which can improve the detection performance. This design choice can effectively address challenge C4.

5 Knowledge Transfer-Based Semi-Supervised Federated Learning Framework

In this section, we describe a practical problem of identifying potential risks related to malware infection in the cloud environments under the FL setting, where clients could not provide the ground truth labels. To address this limitation, we extend the traditional federated learning to the semi-supervised paradigm, and establish a semi-supervised FL framework (FedMalDE) that represents the real-world scenario for malware detection. We first introduce the detection procedures, followed by the details of the algorithm design.

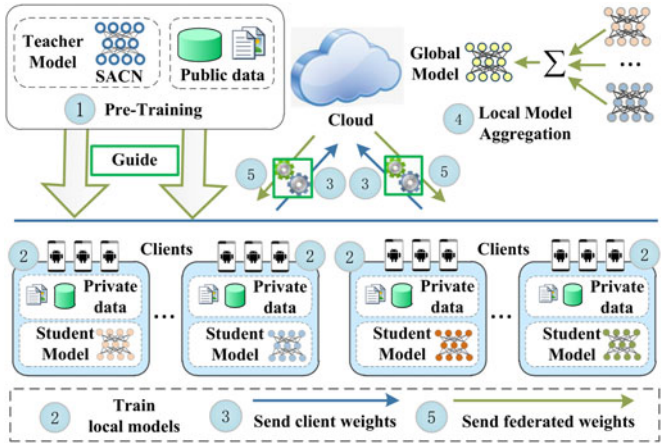


Fig. 3. The overall flow of the proposed FedMalDE.

5.1 Semi-Supervised Federated Learning Process

Under a semi-supervised federated learning setting, we consider a common scenario where the labeled dataset D_T is only available at the cloud, and the unlabeled dataset D_{STU} is privately spread over K clients. Each client $cli_k \in Cli$ locates in different databases consisting of $N_s^{cli_k}$ samples $\{x_i^{cli_k}\}_{i=1}^{N_s^{cli_k}}$. As a result, with the labeled and unlabeled datasets, D_T and D_S, we perform semi-supervised federated learning. In our case, the public labeled data D_T is used to pre-train a teacher model. Afterwards, the teacher model produces high-quality pseudo-labels for users' private data. During the learning process, each client trains its own model locally, and fine-tunes it on limited labeled data. Then, the cloud aggregates the local models from clients, and updates them in a federated manner. This circumvents the need for data exchange between the cloud and the clients, while ensuring privacy for users' data. The learning process is detailed in Fig. 3 and Algorithm 1.

- *Step 1.* We use the labeled public dataset D_T on the cloud to pre-train a teacher model TEA_{model}, as described in Algorithm 1 (Line 1-2).
- *Step 2.* At this stage, we randomly select m out of a total of K idle clients to participate in the FL process as described in Algorithm 1 (Line 3-8). Under the setting of semi-supervised learning, each of m clients downloads the pre-trained teacher model TEA_{model} that assigns a pseudo-label $j \in [c]$ to each sample x_i : $n_j(x_i) = |\{TEA_{model}(x_i) = j\}|$, where c represents the number of classes in our task. Moreover, the teacher model TEA_{model} gives the prediction for the input x_i.
- *Step 3.* At each communication round $r \in R$, each client cli initializes a student model Stu_{model} with the weights w_{tea} of the teacher model, and then trains the student model Stu_{model} to minimize loss $\mathcal{L}_S(w_{cli})$ on the corresponding private sub-dataset $D_s^{cli_k}$ with pseudo-labels, as described in Algorithm 1 (Line 10-25). At this stage, clients keep their local data private, as only the model parameters of the

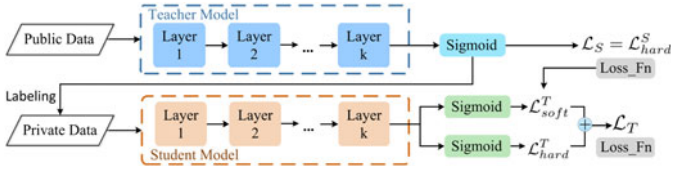


Fig. 4. The knowledge transfer mechanism.

learning model are shared with the central cloud. Specifically, the total communication round R is set to 35.

- *Step 4.* Next, we upload all clients' weights to the cloud. Specifically, to transfer information from teacher to student, we use a knowledge transfer mechanism to enable knowledge matching between the teacher and the students, which guides the clients to update their model weights.
- *Step 5.* The cloud dynamically updates the global model by aggregating the learned weights $w_{cli} \leftarrow \sum_{cli=1}^{CLI_r} \frac{n_{cli}}{n} w^{cli}$ to create a new updated global model. Note that in Algorithm 1 (Line 26-27), the teacher model is also participated in this process, which assists the cloud to aggregate information from student models, i.e., $w_{global} \leftarrow \text{Aggregation}(\{w_{cli}, w_{tea}\})$.
- *Step 6.* The updated global model is sent to all its clients. After that, each client completes its federation round.

Training student models in a semi-supervised way makes better use of the users' private data. The underpinning idea is to leverage the teacher model to learn distribution priors, guiding student models to update their weights. The student models learn to accurately mimic the output of the teacher model, thereby improving the performance of FL.

5.2 Knowledge Transfer Mechanism

In this part, we develop a knowledge transfer mechanism, enabling the federated model to benefit from both public labeled data and private unlabeled data, as illustrated in Fig. 4. To improve the effectiveness of knowledge transfer, we integrate the class probability information generated by the teacher into the calculation of the loss function of the students, which can derive a robust detection model.

5.2.1 Teacher Model

The teacher model is trained with the public data, and its final output layer typically produces the classification probabilities by using a sigmoid function that converts the logit Z_T, computed for each class into a probability $P_{(T,i)}$, where $P_{(T,i)} = \frac{1}{1+exp^{Z_{(T,i)}}}$. Specifically, the Sigmoid function can be used to produce high-quality pseudo-labels (hard targets) for users' private data. Next, we use the cross-entropy as the loss function $\mathcal{L}_T$

$$\mathcal{L}_T = \mathcal{L}^S_{hard}(W_T; P_{(T,i)}) = -\sum_{tea=1}^{N_{TEA}} \big(Y_{(T,i)} \log P_{(T,i)} + \big(1 - Y_{(T,i)}\big) \log\big(1 - P_{(T,i)}\big)\big), \quad (1)$$

where W_T is the parameters of the teacher model, and $Y_{(T,i)}$ is the ground truth.

Algorithm 1. Semi-Supervised Federated Learning

1: $TEA_{model} \leftarrow RunCloudTrain(D_T)$
2: Initialize all clients with weights w_0;
3: **for** each client $cli \in Cli$ **in parallel do**
4: $w_{tea} \leftarrow$ (weights of teacher models)
5: $P_{(T)} \leftarrow TEA_{model(cli)}$
6: $PseudoLabels_{cli} \leftarrow P_{(T)}$
7: $\mathcal{L}_T = \mathcal{L}^T_{hard}(W_T; P_{(T)})$
8: **end for**
9: $w_{global} \leftarrow w_{tea}$
10: **for** each round $r = 1, 2, \ldots, R$ **do**
11: $K \leftarrow$ (number of clients)
12: $C \leftarrow$ (fraction of clients randomly selected per round)
13: $m \leftarrow max(C \times K, 1)$
14: $Cli \leftarrow$ (random set of m clients)
15: **for** each client $cli \in Cli$ **in parallel do**
16: $\mathcal{B} \leftarrow$ (split D^{cli}_S into batches of size B)
17: $\mathcal{P} \leftarrow$ (split $PseudoLabels_{cli}$ into batches of size P)
18: **for** each local epoch $e = 1, 2, \ldots$ **do**
19: **for** batch $b \in \mathcal{B}, p \in \mathcal{P}$ **do**
20: $P_{(S)} \leftarrow RunClientTrain(b_{cli}, p_{cli})$
21: $\mathcal{L}_S = \alpha \mathcal{L}^S_{hard}(W_S; P_{(S)}) + \beta \mathcal{L}^S_{soft}(W_S; (P_{(T)}, P_{(S)}))$
22: $w_{(cli,r+1)} \leftarrow w_{(cli,r)} - \eta \bigtriangledown \mathcal{L}_S(w_{(cli,r)}; b_{(cli,r)})$
23: **end for**
24: **end for**
25: **end for**
26: $w_{(cli,(r+1))} \leftarrow \sum_{cli=1}^{CLI_r} \frac{n_{cli}}{n} w^{cli}_{r+1}$
27: $w_{(global,(r+1))} \leftarrow \text{Aggregation}(\{w_{(cli,(r+1))}, w_{tea}\})$
28: **end for**

5.2.2 Student Model

Analogously, suppose stu_k be a student model with output logit vectors Z_S and class probability $P_{(S,i)} = \frac{1}{1+exp^{Z_{(S,i)}}}$. Then, we train the student model with the sigmoid function to produce the hard targets. The loss function $\mathcal{L}^S_{hard}(W_S; P_{(S,i)})$ is defined as follows:

$$\mathcal{L}^S_{hard}(W_S; P_{(S,i)}) = -\sum_{stu=1}^{N_{STU}} \big(Y_{(S,i)} \log P_{(S,i)} + \big(1 - Y_{(S,i)}\big) \log\big(1 - P_{(S,i)}\big)\big). \quad (2)$$

Specifically, the ground truth $Y_{(S,i)}$ is given by the teacher model, as described earlier. It can make students mimic the output of the teacher model. However, when the teacher model generates pseudo-labels for users, the class probabilities are forced to be converted to one-hot labels, resulting in the inevitably lost class probability information. To remedy this limitation, we integrate the class probability generated by the teacher into the calculation of the loss function. Then, we define an additional loss function $\mathcal{L}^S_{soft}(W_S; (P_{(T,i)}, P_{(S,i)}))$ as follows:

$$\mathcal{L}^S_{soft}(W_S; (P_{(T,i)}, P_{(S,i)})) = -\sum_{stu=1}^{N_{STU}} \big(P_{(T,i)} \log P_{(S,i)} + \big(1 - P_{(T,i)}\big) \log\big(1 - P_{(S,i)}\big)\big), \quad (3)$$

where $P_{(T,i)}$ and $P_{(S,i)}$ are the class probabilities of the teacher model and the student model, respectively. We train the student model to minimize the loss function

$$\mathcal{L}_S = \alpha \mathcal{S}^S_{hard}\big(W_S; P_{(S,i)}\big) + \beta \mathcal{L}^S_{soft}\big(W_S; \big(P_{(T,i)}, P_{(S,i)}\big)\big), \quad (4)$$

where α and β are the tunable parameters, which can control the student's access to the teacher, so that the student's exposure to the teacher's knowledge can be meaningfully bounded. In this paper, α and β are set to 0.5 and 1, respectively.

6 Sliced Subgraph Based Malware Detection

In this section, a graph-based semi-supervised federated learning framework (FedMalDE) is proposed to decentralize a malware detection algorithm and respect the user's privacy. We expect FedMalDE can benefit greatly from the domain knowledge insights through specific algorithm. Specifically, the FedMalDE consists of four main stages. To begin with, we use a disassembler to extract a set of subgraphs (SG) from the Dalvik code for each malware. Then, we proposed a sliced subgraph (SSG) based representation method to model the program semantics and structure. Moreover, we use the word embedding method to inject the SSG into the embedding vector space to obtain better feature representations. In this process, each of them is associated with a corresponding dense real-valued vector. Finally, a specially designed subgraph aggregated capsule network (SACN) is proposed to aggregate domain knowledge from complex call graphs.

6.1 Sliced Subgraph Based Representation

Given an application, static analysis is performed to transform the program codes into a graphical representation to present the behaviors and structures. Our goal is to predict whether a given application is malware. We first give the basic definition of Function Call Graph (FCG) before introducing the sliced subgraph.

6.1.1 Function Call Graph (FCG)

The Function Call Graph (FCG) can be derived from an Android application by code analysis, which reflects the program behaviors.

Definition 1 (FCG). *Define a FCG as $G = (\mathcal{V}, \mathcal{E})$ where V and $\mathcal{E}$ represent the set of N nodes and M edges, respectively.*

- *Each node $v_i \in \mathcal{V}$ corresponds to a function invoked by a given app.*
- *Similarly, each edge $e_{i,j} \in \mathcal{E}$ connects two nodes v_i and v_j, which indicates the calling relation between the caller function v_i and the callee function v_j.*

By extracting the callers and the callees from the Dalvik code of an given app, FCG can describe the necessary structural information.

Analysis. Although some studies [15] have proved that FCG is effective for malware detection, there are still some problems for applying it in practice based on the following observations:

Observation 1. *Clearly, analyzing the entire FCG is inefficient and time-consuming since thousands of nodes (methods) can be found in the FCG.*

Observation 2. *FCG is seldom focused on program semantics at the class level, but rather, at the method level only. Although more fine-grained local features can be extracted, FCG ignores the global information about the malicious behaviors.*

6.1.2 Sub-Graph (SG)

To solve these challenges mentioned above, we develop a sub-function call graph-based representation approach. We slice the FCG into n subgraphs $\{SG_1, SG_2, \ldots, SG_n\}$. Moreover, instead of constructing FCG at method level, the SG is constructed at class level [34]. In this setting, the whole program can be represented according to the correlations between classes. This strategy allows us to describe the program semantics from a global perspective. Specifically, each node (class) extracts the features from all its methods in the class as node attributes, allowing us to benefit from local information as well. The following provides the basic notation of SG.

Definition 2 (SG). *Define a SG as $G' = (\mathcal{V}, \mathcal{E}, \mathcal{X}_\mathcal{V}, \mathcal{X}_\mathcal{E}, \mathcal{A})$ where V and $\mathcal{E}$ represent the set of N nodes and M edges, respectively. SG is a sub-graph in FCG.*

- *Define a $\mathcal{X}_\mathcal{V} \in R^{N \times D_1 \times D_2}$ as the node attribute matrix, where N indicates the number of nodes, D_1 indicates the number of node attributes, and D_2 indicates the dimension of the node attribute. Each node v_i is associated with an attribute representation $\mathcal{X}_{\mathcal{V},i} \in R^{1 \times D_1 \times D_2}$.*
- *Similarly, the $\mathcal{X}_\mathcal{E} \epsilon R^{M \times D_3 \times D_4}$ is defined as the edge attribute matrix.*
- *Define a $\mathcal{A} \in R^{N \times N}$ as the adjacency matrix, where $A_{i_1,i_2} = 1$ if $e_{i_1,i_2} \in \mathcal{E}$. Otherwise, $A_{i_1,i_2} = 0$.*

6.1.3 Sliced Subgraph (SSG)

To identify malware, we distilled program semantics and structural information into the sliced subgraph (SSG) representation. The malicious parts are more likely to be buried in these complex functions and call sequences. We seek to exploit a combination of multiple SGs. Intuitively, the combination can represent the program semantics of the entire FCG. For knowledge refinement, we merely focus on the top K subgraphs as the candidate subgraphs, which reduces the dimensionality, making the problem more tractable.

Definition 3 (SSG). *Define a SSG as $G''=\{G'\}^k$, where G' represents the SG.*

Specifically, the SSG is a set of sub-function call graphs, which can reflect the program semantics and the call relations among the classes, as shown in Fig. 5.

6.2 Generation of SSG

The entire application can be rendered by a combination of SGs. To get the best of both worlds, we extract the program semantics and graph structure information from an app for constructing node attributes and edge attributes, respectively. An example of the graph representation is shown in

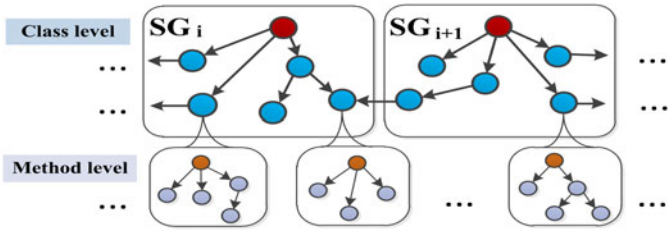


Fig. 5. The generation of sliced sub-graphs.

Fig. 6. We consider the following questions to better understand the construction of SSG.

Q1: How to utilize program semantic?

To depict the program semantics, the features of all methods in a class are extracted as the class (node) attributes. The key idea is to progressively aggregate the fine-grained features to obtain a high-quality node representation.

6.2.1 Node Attribute

In the feature extraction, we focus on features that help distinguish malware from benign apps. After the observation and analysis of the features, the irrelevant information is eliminated, and the following four types of features are extracted as the node attributes:

Parameter Features (PF): Each method has at least one parameter. Thus, we record the number of the parameters for each method, which is defined as PF_{par}. Moreover, we also extract a total of 4 features, including the length of method PF_{len}, the values of opcode PF_{op}, the number of registers PF_{reg}, and whether the return type is void PF_{void}

$$PF_i = \{PF_{par,i}, PF_{len,i}, PF_{op,i}, PF_{reg,i}, PF_{void,i}\}, \tag{5}$$

where PF_i is the parameter features extracted from i-th method in a class.

Sensitive API Features (SAF): We extracted all the API calls (SAF_{api}) contained in a method, if they exist. Then, we extract the sensitive API calls (SAF_{sapi}) from SAF_{api} as a special feature set, as they may lead to malicious behaviors. The set is defined as follows:

$$SAF_i = \{SAF_{api,i}, SAF_{sapi,i}\}. \tag{6}$$

Instruction Features (IF): Based on a scenario in which a method uses a set of instructions to access the registers, we extracted the instructions as the method features, which include the number of data types (e.g., *const-string*) and arrays, the instructions of move, jump (e.g., *if* and *switch*), compare (e.g., *cmp*), exception (e.g., *throw*), and invoke (e.g, *invoke-static*), and the operations of instances (e.g., *new-instance* and *check*) and fields (i.e., *sget* and *sput*)

$$IF_i = \{IF_{dt,i}, IF_{arr,i}, IF_{mov,i}, IF_{jum,i}, IF_{com,i}, IF_{exc,i}, IF_{inv,i}, IF_{i,ins,i}, IF_{fi,i}\}. \tag{7}$$

Modifier Features (MF): Access modifiers are used in setting accessibility values. Each access modifier controls the access level for classes and its members. Thus, the access modifiers (including *public* MF_{pub}, *protect* MF_{pro}, and *private* MF_{pri}) are treated as the modifier features, and all other non-access modifiers (such as *abstract* and *final*) are considered as others MF_{other} as follows:

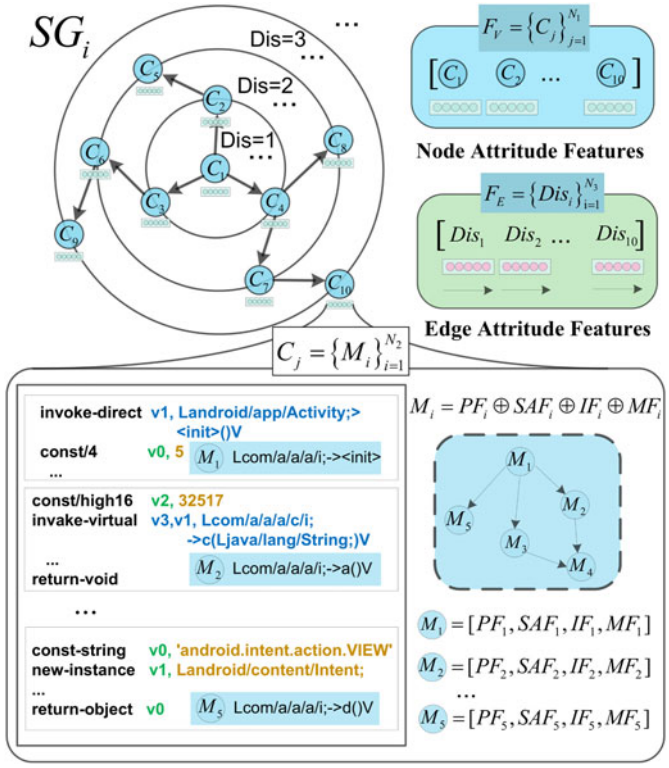


Fig. 6. Details of graph embedding.

$$MF_i = \{MF_{pub,i}, MF_{pro,i}, MF_{pri,i}, MF_{other,i}\}. \tag{8}$$

Then, the node (class) attribute features $F_{\mathcal{V}}$ can be represented by:

$$M_i = PF_i \oplus SAF_i \oplus IF_i \oplus MF_i, \tag{9}$$

$$F_{\mathcal{V}} = \{C_j\}_{j=1}^{N_1} = \{\{M_i\}_{i=1}^{N_2}\}_{j=1}^{N_1}, \tag{10}$$

where N_1 is the number of classes, and N_2 is the number of methods in a class.

Q2: How to utilize graphic structural information?

We consider that the structural information can be depicted by the execution paths between classes, which reflect the program logic. For the utilization of the structural information, we captured all execution paths according to the class call relations as the basis of edge attributes.

6.2.2 Edge Attribute

For each subgraph, the root class in the hierarchy can be regarded as the central node $\mathcal{V}_u$. We use $\mathcal{N}_{(\mathcal{V}_u)}$ to denote the set of its neighbors. To simplify things, the execution paths from $\mathcal{V}_u$ to its k-hop neighbors are used to describe the hierarchical structures. Moreover, each path $(\mathcal{V}_u \rightarrow \mathcal{V}_v)$ has a distance $Dis_{(\mathcal{V}_u,\mathcal{V}_v)}$. If a neighbour node $\mathcal{V}_v$ of $\mathcal{V}_u$ is included at the k-th hop, we will give a value k to $Dis_{(\mathcal{V}_u,\mathcal{V}_v)}$. Finally, the edge attribute features can be represented by:

$$F_{\mathcal{E}} = \{Dis_i\}_{i=1}^{N_3}. \tag{11}$$

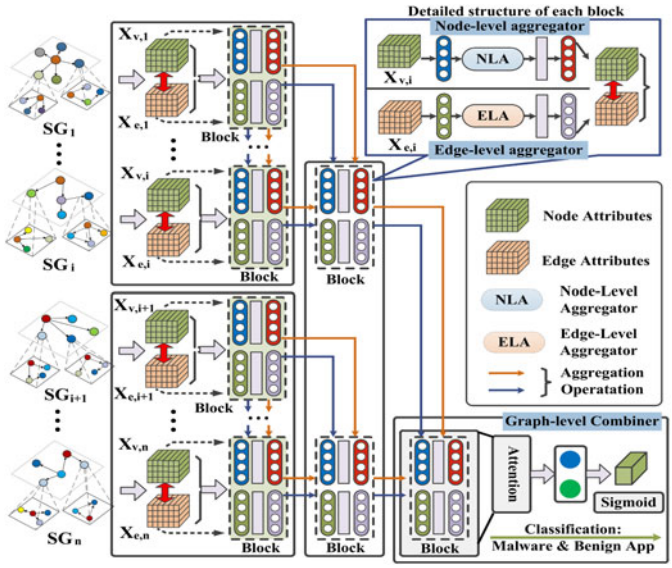


Fig. 7. The overall architecture of subgraph aggregated capsule network. The model consists of multiple blocks. Each block contains two modules, the node-level aggregator (Section 6.4.1), and the edge-level aggregator (Section 6.4.2). Lastly, the graph-level information is extracted by a graph-level combiner (Section 6.4.3).

The edge attributes can sufficiently discriminate nodes of different distances to the target central node.

6.3 Graph Embedding

The outcomes of previous steps are a set of subgraphs as well their corresponding node attributes and edge attributes. Recall that the nodes in a subgraph are actually classes, and each node attribute is associated with a set $F_{\mathcal{V}} \in R^{D_1}$ (where D_1 represent the set size), which integrates the program semantics of all methods in the corresponding class. Similarly, the edge attributes integrate the graph structure information.

Q3: How to impove the quality of graph represention?

In this part, our task is to embed the node attributes and edge attributes into a low-dimension vector space by the embedding method, aiming to better construct the program semantics and the structure information. Moreover, each feature is converted into a embedding vector $x_v \in R^{D_2}$, where D_2 represent the feature dimension. Finally, the node attribute matrix can be represented by $\mathcal{X}_{\mathcal{V}} \in R^{N \times D_1 \times D_2}$, where N represent the number of nodes. Similarly, we can have the edge attribute matrix $\mathcal{X}_{\mathcal{E}}$.

6.4 Subgraph Aggregated Capsule Networks

In this subsection, we describe the details of our model SACN. The SACN takes L subgraphs $G'' = [G'_1, G'_2, \ldots, G'_L]$ as the inputs, which aggregates the fine-grained information at multiple levels. The overall flow of SACN is illustrated in Fig. 7. During the feature aggregation, we consider multiple domain knowledge through the following three modules: (1) a node-level aggregator that learns the program semantics from the node attributes; (2) an edge-level aggregator that identifies the graphic structural information from the edge attributes; (3) a graph-level combiner that recombines the above learned features to obtain unified graph embedding. Finally, a sigmoid function is applied to output the prediction.

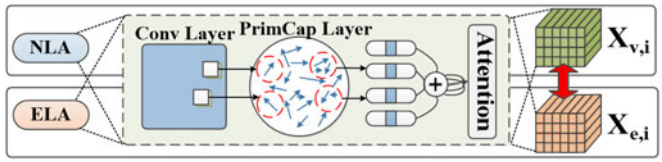


Fig. 8. Details of the node-level aggregator and the edge-level aggregator.

6.4.1 Node-Level Aggregator

Given a SSG G'' consisting of $L_{\mathcal{V}}$ subgraphs SG G', we try to learn representation of the node-level features. The role of the node-level aggregator is to properly process the subgraphs so that different fine-grained information can be mapped to different representations.

Given a subgraph G' with a node sequence (i.e., $G' = \mathcal{X}_{\mathcal{V}} = \{\mathcal{X}_v\}^{T_v}$) where each node $\mathcal{X}_v$ in the sequence can be presented by multiple dimensions such as node attributes. Then, we slice the node sequence into n_v subsequences N_v with an equal length $t_v = T_v/n_v$ (i.e., $\{N_{v,1}, N_{v,2}, \ldots, N_{v,n_v}\}$), and each subsequence can be represented by:

$$N_{v,p} = [\mathcal{X}_{v,(p-1)\times t_v+1}, \mathcal{X}_{v,(p-1)\times t_v+2}, \ldots, \mathcal{X}_{v,p\times t_v}]. \tag{12}$$

By repeating the operation l times, we can have a set of subsequences of equal length on the l layer. The length of a subsequence is defined by $t_{v,l} = T_v/n_v^l$, and the number of the minimum subsequences is defined by n_v^l. Then, we use a node-level aggregator (NLA) for each subsequence. By doing this, the fine-grained information can be aggregated on multiple levels. Formally

$$O^l_{\mathcal{X}_{v,i}} = NLA(\mathcal{X}_{v,i})^l, \tag{13}$$

where the size of $O^l_{\mathcal{X}_v}$ is $t_{v,l} \times h$, with h being the number of hidden states. With the parallel calculations, the input is changed to $\{\mathcal{X}^l_{v,1} || \mathcal{X}^l_{v,2} || \cdots || \mathcal{X}^l_{v,n_v^l}\}$, where the $\mathcal{X}^l_{v,i}$ is the i-th input of NLA. Fig. 8 shows the details of NLA. Specifically, for i-th input $\mathcal{X}^l_{v,i}$, it is first fed into a standard convolutional layer to generate β feature maps F in the l-th NLA layer, which can be represented as $F = [F_1, F_2, \ldots, F_\beta] \in R^{(n-C_1+1)\times\beta}$, where F_j represent the j-th column feature map, C_1 is the size of convolutional filter. Then, we use the primary capsule to combine the instantiated parts by converting the scalar-output to vector-output, so that the instantiated parameters for each feature can be preserved, which represent the intensity of activation. In this way, the model learns the feature attributes from a different viewpoint and records some details of the instantiated parts.

Let $W^p \in R^{C_2 \times d}$ be the filter, where C_2 is the filter size and d is the dimension of the capsule. $F_j \in R^\beta$ denotes each vector generated by the convolution operation. A column-list of capsules can be calculated by:

$$P = (W^p)^T \{F_j\}_{j=1}^{n-C_2+1} \in R^{(n-C_2+1)\times d}. \tag{14}$$

Moreover, each capsule $P_j \in R^d$ in P can be obtained by $P_j = g(s_j)$, where $s_j = (W^p)^T F_j + b_j$, and g represents the nonlinear squash function that can be calculated by:

$$g(s_j) = \frac{||s_j||^2}{1 + ||s_j||^2} \frac{s_j}{||s_j||^2}. \quad (15)$$

Suppose there are C filters. For $j \in 1, 2, \ldots, C$, the capsule feature maps can be obtained as

$$V = [P_1, P_2, \ldots, P_C] \in R^{(n-C_2+1)\times C\times d}. \quad (16)$$

Moreover, we apply an attention mechanism to enhance the representations of important capsules.

For the SACN, each of the higher layers can not only learn the features of the current layer but also aggregate the low-level features from the previous layer. Specifically, the NLA in the SACN is set to three layers as shown in Fig. 7 for clarity. In practice, by stacking multiple layers, the aggregator can be easily extended to deeper networks, which can aggregate the fine-grained information at multiple levels.

6.4.2 Edge-Level Aggregator

In previous studies [15], [16], [17], researchers have only focused on the representations of nodes, ignoring the graphic structural information, so that their modeling capabilities are imitated. In the face of this challenge, we design an edge aggregation module to automatically learn the subgraph structure information that depicts the execution paths between classes.

Given a subgraph G' consisting of $L_{\mathcal{E}}$ edges (i.e., $G' = \mathcal{X}_{\mathcal{E}} = \{\mathcal{X}_e\}^{T_e}$). By slicing the edge sequence, we can have n_e subsequences N_e with an equal length $t_e = T_e/n_e$, and each subsequence can be represented by:

$$N_{e,p} = [\mathcal{X}_{e,(p-1)\times t_e+1}, \mathcal{X}_{e,(p-1)\times t_e+2}, \ldots, \mathcal{X}_{e,p\times t_e}]. \quad (17)$$

Moreover, the n_e subsequences $\{\mathcal{X}_{e,1}||\mathcal{X}_{e,2}||\cdots||\mathcal{X}_{e,n_e}\}$ can be used as the inputs of the edge-level aggregator (ELA) at each step through the parallel calculations. After performing operations similar to node-level aggregator (NLA), we can have a set of subsequences of equal length on the l layer. Formally

$$O^l_{\mathcal{X}_{e,i}} = ELA(\mathcal{X}_{e,i})^l. \quad (18)$$

6.4.3 Graph-Level Combiner

In addition to the above two aggregators, for better classification, we also extract the graph-level information. In this module, the output capsules of the two aggregators from the previous layer are flattened into a list of capsules, and combined by an internal product layer. Then, each capsule is connected to a list of low-layer capsules via a weight matrix that is used to produce the final capsule. Instead of the traditional pooling algorithms, dynamic routing is used to conduct an efficient layer-to-layer information delivery among capsules. The output of the graph-level combiner (GLC) can be given by:

$$O_{glc} = GLC([O_{\mathcal{X}_{\mathcal{V}}}, O_{\mathcal{X}_{\mathcal{E}}}]), \quad (19)$$

where "[]" represents the concatenation operation. For knowledge distillation, an attention mechanism is further used to enhance the representations of important capsules. In order to obtain the weight distribution, the normalized importance weight a_i is first calculated by:

$$a_i = \frac{\exp(u_i^T u_w)}{\sum_i \exp(u_i^T u_w)}. \quad (20)$$

where $u_i = tanh(W_w h_i + b_w)$. Then, the final vectors can be obtained by $v = \sum_i a_i h_i$. Moreover, the whole calculation process can be formalized as

$$O_{Att} = Att(O_{glc}). \quad (21)$$

Finally, the sigmoid layer is added after the attention layer to give the classification probabilities, and the cross entropy is used as the loss function.

7 Evaluation

In this section, we present data preprocessing and implementation details. We implement our SACN model by using the Keras platform, and use word2vec to learn graph embeddings.

Federated Learning Setup: To evaluate our method, we empirically evaluated FedMalDE by simulating a single cloud, and multiplying edges and clients (with "their part" of the data). In each communication round, we randomly selected only m clients to participate in the training. In addition, a "single model" is also evaluated (a single cloud with all the labeled data). In the following sections, we will show how FedMalDE performs with different configurations, as well as its evolution across rounds with respect to the different baseline learning models in FedMalDE.

7.1 Datasets and Evaluation Metrics

7.1.1 Datasets

For a comprehensive assessment, we evaluated FedMalDE on four benchmark datasets. DREBIN [35] is an Android malware dataset that contains 5,565 samples. We collected 5,565 benign apps from a variety of sources for further experiments. MalDroid (MD) [36] is an Android malware dataset collected from December 2017 to December 2018. In our experiments, 12,008 malware samples and 12,008 benign apps are used. Moreover, AndroZoo (AZ) [37] dataset for 2010 through 2020 consists of all applications from respective years. Along with these datasets, we collected 21,871 malware samples from VirusShare [38] to build the VS dataset. In addition, to balance this dataset, we also collected 21,871 benign samples from various sources. Specifically, we also evaluated FedMalDE with unbalanced data in Section 7.2.3. In our experiment, 40% of the samples being randomly selected as a labeled training set which is used to train the teacher models, 40% of the samples as a training set with pseudo-labels generated

TABLE 2
Main Datasets Used in Our Evaluation Studies

Datasets	#Malware	#Benign apps	#Total
DREBIN	5,565	5,565	11,130
MD	12,008	12,008	24,016
VS	21,871	21,871	43,742
AZ	31,673	31,673	63,346

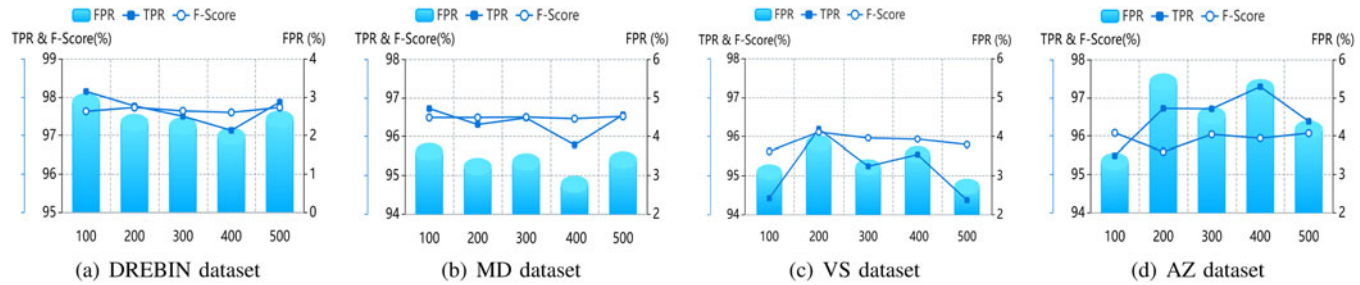


Fig. 9. Number of clients contributing in federated learning.

by the teacher ensemble which is used to train the student models, and the remaining as the test set. Note that this evaluation study did not involve any re-sampling o avoid biases or overfitting. Table 2 lists the descriptions of the four datasets.

7.1.2 Experimental Setup and Parameters Setting

The setting of model parameters has an important impact on detection results. As mentioned previously, our detection model is formed by merging three modules, a node-level aggregator, an edge-level aggregator, and a graph-level combiner. Each module mainly consists of a capsule layer and an attention layer. SACN can transmit important information through a multiple-layer structure. Some important training details are as follows: (i) the dimension of word embedding is set to 150; (ii) the number of the capsule is set to 64, and each capsule has 32 dimensions; (iii) the sigmoid layer is used as the output of our detection model, which gives the classification probabilities; (iv) the batch size is set to 32; (v) the dropout rate is set to 0.5; (vi) we use Adam with learning rate $\alpha = 0.001$ to optimize the training process, and (vii) the back propagation is used minimize the loss, where the loss function is set to cross entropy.

7.1.3 Evaluation Metrics

For evaluating the classification results, some common machine learning performance evaluation metrics are used to quantify numerically the performance of the classifier, which include false positive rate (FPR), true positive rate (TPR), ROC curve, and F-score. FPR measures the rate at which the benign samples are incorrectly classified as malware, thereby causing false alarms. TPR corresponds to the proportion of malicious apps that are correctly reported, which is also known as recall or detection rate. The Receiver Operating Characteristic (ROC) curve illustrates how good our model is in classifying classes based on FPR and TPR. The F-score is the harmonic mean of precision and recall. Our goal is to achieve high values for F-score and TPR while obtaining the minimum FPR.

7.2 Performance Stability

7.2.1 Effects of Client Number

We consider that there will be many federation rounds to train the local models. Therefore, we conducted an additional study to test the classification performance of federated learning with the different numbers of clients (ranging from 100 to 500) contributing to the training of the federated learning, as shown in Fig. 9. Specifically, each client trains its local model. The results show that the proposed SACN can achieve a satisfactory result without too much client involvement, and our model is insensitive to the number of clients. It indicates that it is beneficial to combine semi-supervised learning with federated learning. For efficiency, we only randomly select a fraction C of clients at the beginning of each round. Fig. 10 shows that the return will diminish when the number of clients exceeds a certain point.

7.2.2 Performance Comparison for the Proportion of Labeled Public Data

We also conducted an additional study to investigate the impact of the number of labeled public training sets used for training a teacher model. Fig. 11 illustrates the accuracy of FedMalDE with respect to the number of labeled public training sets. We evaluate our model on five proportions of the labeled training data on the four benchmark datasets. For example, in the case of a small proportion, we split the DREBIN dataset into labeled public training, unlabeled private training, and testing sets in the proportion of 3:6:1. As described in Section 5, FedMalDE pre-trains a teacher model with labeled public data, and then generates pseudo-labels for user data. Intuitively, our model can fit the data distribution better and achieve better performance, when more

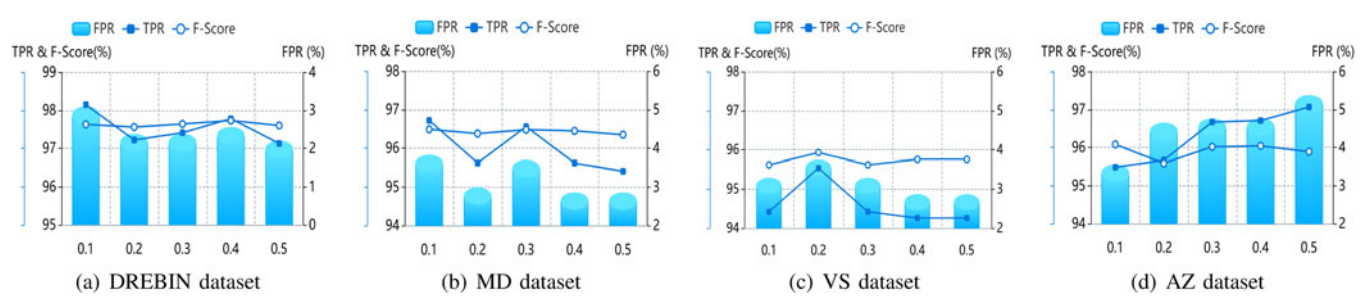


Fig. 10. Fraction of clients randomly selected per round of federation.

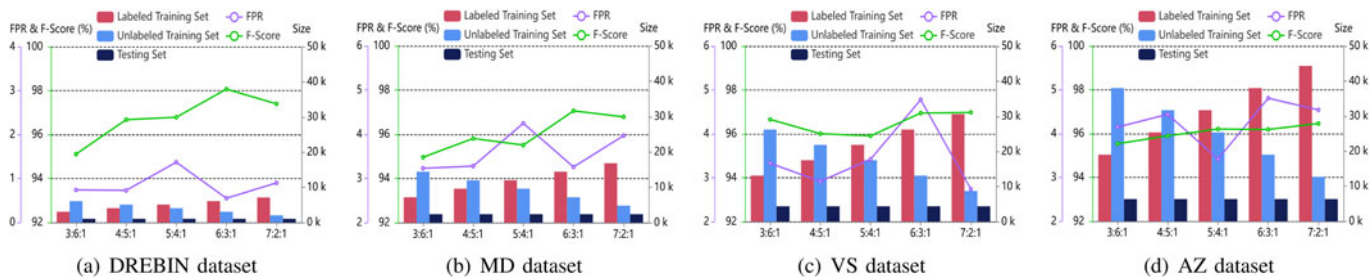


Fig. 11. Performance on different labeled data proportions.

labeled samples are added. This is established in Fig. 11, which shows that the F-score increases significantly with increasing labeled data, and then stabilizes. In addition, as expected, we observe that FedMalDE exhibits worse performance when the number of samples in the labeled training set is small. This is because the teacher model does not learn enough domain knowledge due to the data scarcity, which gives wrong guidance to the student model, leading to misprediction of samples in the test set. However, somewhat surprisingly, we noticed that FedMalDE still maintains a high F-score with not too much labeled data. This suggests that, in some use cases, the proportion of labeled data may have a relatively low impact on the performance of FedMalDE.

7.2.3 Performance Comparison for the Proportion of Malware and Benign Samples

To further study the generalization capability of our system, we consider the impact of the number of malicious and benign samples in different proportions on classification performance, and all samples are randomly distributed to each client. There should be more data labeled as benign, as users typically install more clean apps than malicious apps. Assuming that the adversaries can control a large number of devices, as is highly likely in real-world situations, it will mean that the distribution of the data participating in training of the local models could be changed, resulting in the local models being poisoned. It can be seen from the results shown in Fig. 12 that as the number of malware samples increases, the F-score drops slightly. Conversely, the FPR has a trend of increase. This means that malware has not been misclassified as benign apps. Moreover, this may be because the increase in the number of malicious samples allows the model to learn more information, so as to avoid being affected by changes in the proportion of malware and benign samples. This finding agrees with our speculation on generalization: the proposed SACN has a strong classification ability for unbalanced data, which corroborates the robustness merits of our method.

7.3 Parameter Sensitivity Analysis

In this experiment, we perform sensitivity analysis to some main experiment parameters in FedMalDE. Due to the limited space, we only reported the parameter analysis results on the MD dataset, as shown in Fig. 13. In the following, we provided the parameters commonly used in our proposed model. Then, our best performing setup is used to fine-tune a final federated model FedMalDE. The robustness of our model is evident by the difference in performance metrics.

7.3.1 Dimension of the Embedding

We discuss the effect of dimensions of the embedding. We selected the values [30, 50, 100, 150, 200] to investigate its influence on the federated model. Fig. 13a shows that as the dimension of the embedding increases, the model fits the data distribution better. Because of that the increase in the embedding dimensions provides our model access to more semantic information.

7.3.2 Convolutional Parameter

We investigate the effect of different sizes of kernel and filter in the convolutional capsule network layer. From Figs. 13b and 13c, we can see that as the sizes of kernel and filter grow, the classification performance of our system raises steadily. This is not surprising since the model can only learn less information when the sizes of kernel and filter are small. Generally, we can conclude that a modest increase in the sizes of kernel and filter would improve the classification performance.

7.3.3 Effects of Capsule Network

We evaluate the classification performance for different capsule numbers and dimensions involved in building the

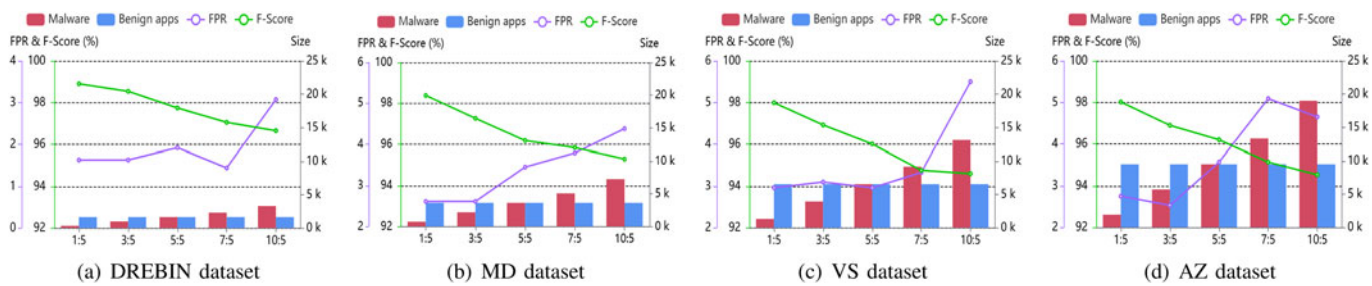


Fig. 12. The comparative classification performance in various ratios of benign to malware samples.

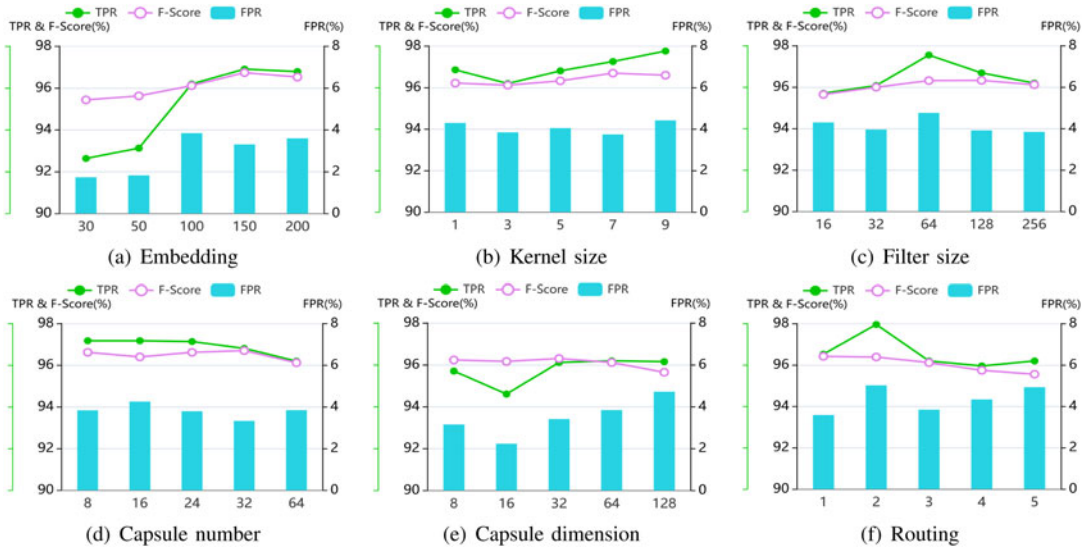


Fig. 13. Parameter sensitivity of FedMalDE.

federated model. From Figs. 13d, 13e, and 13f, we can see that our model is insensitive to these three parameters. Specifically, the model performs the worst when the number of capsules, the dimensions of capsules and the number of iterations in the routing process are set to be 64, 128, and 5, respectively. This may be explained by the fact that the capsule network algorithm is easy to converge, which can capture the correlation (e.g., class call relations) among different nodes (classes).

7.4 Centralized SACN versus Federated SACN

In this experiment, we evaluated the classification performance of centralized SACN and federated SACN. Compared with the centralized SACN, the SACN trained in a federated manner may result in a loss of accuracy. For comparison, we specified the number of clients to be 200. Table 3 shows the performance evaluation results. We can see that compared with centralized SACN, the performance of federated SACN on the TPR, FPR, and F-measure is slightly reduced. This small degradation in the performance of the federated SACN is not a concern. It indicates that the federated SACN has a strong classification ability for malware detection. This may be because the node-level aggregator and the edge-level aggregator consider the correlation among various features, thereby learning from multiple angles to the various features of the same threat pattern.

7.5 Comparative Classification Performance

We have used five different classifiers as the baseline learning models in FedMalDE to compare the performance of the proposed SACN. These baselines include Deep Neural Network (DNN) [39], Convolutional Neural Network (CNN) [40], Gated Recurrent Unit (GRU) [41], Long Short Term Memory (LSTM) [42] and Capsule Network (CAP) [43]. Fig. 14 and Table 3 provide the performances of all baselines and our method on four benchmark datasets. It is worth noting that all baseline models are trained under the same semi-supervised FL setting. Fig. 14 and Table 3 present the performance comparison of all baselines and our method on the four benchmark datasets. We can see that all of the compared models perform well on the DREBIN dataset, while performing poorly on the other three datasets. Although DNN, GRU, and

TABLE 3
Effect of Using Federated Learning Comparing to Centralizing Approach

Datasets		Centralized learning						Federated learning					
		DNN	CNN	GRU	LSTM	CAP	Our	DNN	CNN	GRU	LSTM	CAP	Our
DREBIN	TPR	95.37	94.36	96.58	85.85	96.95	**98.24**	94.82	96.30	96.58	90.48	96.30	**97.50**
	FPR	**0.90**	0.90	3.26	12.5	2.71	2.71	**0.63**	2.98	3.35	17.66	2.17	2.26
	F-Score	97.33	96.85	96.69	86.90	97.15	**97.76**	97.20	96.70	96.64	85.91	97.12	**97.64**
MD	TPR	94.11	89.16	93.86	85.19	92.14	**96.89**	95.74	89.16	94.51	40.96	92.18	**96.8**
	FPR	4.21	**1.25**	7.49	1.60	6.14	3.62	6.52	**1.26**	8.50	2.52	6.31	3.73
	F-Score	94.90	88.03	93.05	84.28	92.94	**96.58**	94.48	87.98	92.82	75.47	92.87	**96.48**
VS	TPR	97.24	98.11	96.31	98.21	**98.43**	97.82	96.79	98.14	96.17	**98.34**	97.17	97.01
	FPR	**1.01**	22.48	6.52	11.98	1.42	4.78	**0.98**	22.58	6.43	12.55	1.35	4.12
	F-Score	93.27	86.38	94.79	92.73	91.48	**96.45**	93.20	86.33	94.77	92.47	91.27	**96.40**
AZ	TPR	92.09	89.16	92.37	92.08	88.85	**96.44**	93.01	89.13	91.87	88.30	88.36	**95.75**
	FPR	7.49	9.86	8.16	15.41	7.70	**4.11**	8.33	9.85	7.88	11.02	7.31	**3.93**
	F-Score	92.14	89.63	92.03	87.83	90.50	**96.14**	92.13	89.62	91.95	88.60	90.48	**95.86**

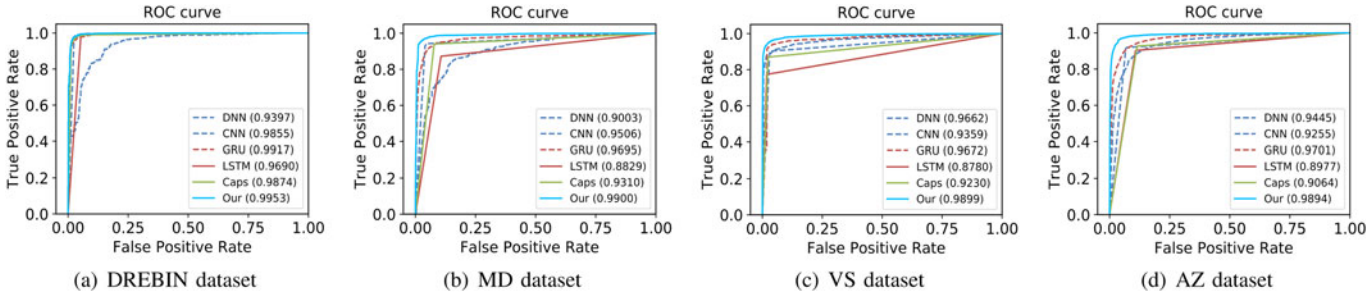


Fig. 14. ROC curve of the SACN versus baselines for malware detection on four benchmark detasets.

CAP are closest to our model SACN in overall performance on the four benchmark datasets, they are not yet comparable to SACN. Moreover, these baseline models are inclined toward accuracy (TPR) rather than recall (FPR), while our method SACN offers a more balanced performance as shown in Table 3. As we expect, under the federated setting, SACN performs consistently well on all datasets and outperforms all baseline models. This is because SACN can model the program semantics and structural information by progressively aggregating the fine-grained subgraph features, which can accurately and efficiently process the data. The superior performance of SACN in the malware detection task reflects its powerful generalization ability.

7.6 Processing Time Evaluation

Another topic of discussion is the efficiency of FedMalDE for detecting arbitrary malware. For a fair comparison, all experiments are run at the same hardware resources. Analyzing a malware can be divided into two phases: feature extraction and classification. For a malware detection system, the bottleneck in efficiency is mainly in the feature extraction stage. We recorded the runtime for processing one sample in the feature extraction stage. All baselines underwent the same feature extraction, and none of them avoided this stage. It takes less than 3 minutes on average to analyze a sample, which is comparable with other state-of-the-art approaches, such as DroidCat [44] (10 min tracing per app) and MamaDroid [45] (mean of 4 min per app).

Additionally, we recorded the runtimes for training a teacher model. As shown in Table 4, our method is slower than some baseline methods, such as CNN and DNN. This is because our model is more complex and it is able to extract more useful information. Apparently, given the substantially superior performance of the FedMalDE over other baseline methods (see Table 3 in Section 7.5 for details), the additional cost incurred by the FedMalDE can be seen to be justified. It is worth noting that our model became optimal after approximately 20 iterations. In practice, however, the cloud only needs to pre-train the teacher model once.

Under the setting of semi-supervised FL, we also measured the runtimes for training a student model on a single device. In typical real-world applications, there are few apps (private data) installed on the user's device. In the training mode, the student models can be parallelized unlike conventional centralized models which handle all data on the cloud, and thus only take a small amount of time to train. For example, a user with 20 apps spends approximately 2 seconds to train a student model in a round of federation. Therefore, this allows us to easily implement on-device learning to retrain a large number of emerging samples in the real world. It is worth noting that the time spent on model inference is almost negligible. For example, the average lies at just 0.1876 seconds for processing one sample. Based on the above facts, we can claim that FedMalDE performs FL with high performance and efficiency.

8 Conclusion

This paper proposes a semi-supervised federated learning framework (FedMalDE) that identifies the potential risks to IoT malware infections in IoT networks, while alleviating data privacy issues on user devices. The FedMalDE explores the use of semi-supervised federated learning for analysis of malware to conduct distributed FL, and it can provide sufficient privacy and robustness guarantees. Besides,

TABLE 4
Execution Time Comparison

Model	Teacher model		Student model		Inference / Testing time		
	#Train samples	Train time (per epoch)	#Single device	Train time (per epoch)	#Testing samples	Testing time	Avg. time (per sample)
DNN	10,000	132 s	20	< 1 s	5000	551 s	0.1102 s
CNN	10,000	133 s	20	< 1 s	5000	528 s	0.1056 s
CAP	10,000	313 s	20	< 2 s	5000	1047 s	0.2094 s
GRU	10,000	2816 s	20	< 11 s	5000	972 s	0.1944 s
LSTM	10,000	2985 s	20	< 11 s	5000	979 s	0.1958 s
Our	10,000	498 s	20	< 2 s	5000	938 s	0.1876 s

considering that the function call graph can reflect the semantics information of the entire program, we incorporate the graph mining technology into this framework to reveal the malicious behaviors, and use a subgraph aggregated capsule network (SACN) to capture various granularity and structural information. The extensive experiments conducted on real-world data validate the effectiveness of FedMalDE. We believe that our work will inspire the community to further explore the implementation of FL in security-related tasks.

There might be room for further FL-based contributions in the field of malware detection. In each federation round, the cloud communicates with only a limited number of devices. However, unreliable communication channels, limited computing resources, and stringent training latency budget hinder the convergence of FL. Future research will focus on exploring new device scheduling and resource allocation schemes to balance the latency per round, as well as the number of rounds required. It is generally believed that centralized-based schemes are considered to be the best in terms of model accuracy. Therefore, future work is needed to improve the accuracy of federated models.

ACKNOWLEDGMENTS

The authors would like to thank the Editor-in-Chief, the Associate Editor, and the reviewers for their insightful comments and suggestions.

REFERENCES

[1] L. Qi, X. Zhang, S. Li, S. Wan, Y. Wen, and W. Gong, "Spatial-temporal data-driven service recommendation with privacy-preservation," *Inf. Sci.*, vol. 515, pp. 91–102, 2020.

[2] A. Salem, Y. Zhang, M. Humbert, P. Berrang, M. Fritz, and M. Backes, "ML-leaks: Model and data independent membership inference attacks and defenses on machine learning models," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2019, pp. 1–15.

[3] L. U. Khan, W. Saad, Z. Han, E. Hossain, and C. S. Hong, "Federated learning for Internet of Things: Recent advances, taxonomy, and open challenges," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 3, pp. 1759–1799, Jul.–Sep. 2021.

[4] J. Feng, L. T. Yang, Q. Zhu, and K.-K. R. Choo, "Privacy-preserving tensor decomposition over encrypted data in a federated cloud environment," *IEEE Trans. Dependable Secure Comput.*, vol. 17, no. 4, pp. 857–868, Jul./Aug. 2020.

[5] L. Lyu *et al.*, "Privacy and robustness in federated learning: Attacks and defenses," 2020, *arXiv:2012.06337*.

[6] S. Sharma, C. Xing, Y. Liu, and Y. Kang, "Secure and efficient federated transfer learning," in *Proc. IEEE Int. Conf. Big Data*, 2019, pp. 2569–2576.

[7] L. Zhao *et al.*, "Shielding collaborative learning: Mitigating poisoning attacks through client-side detection," *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 5, pp. 2029–2041, Sep./Oct. 2021.

[8] L. Zhao, J. Jiang, B. Feng, Q. Wang, C. Shen, and Q. Li, "SEAR: Secure and efficient aggregation for byzantine-robust federated learning," *IEEE Trans. Dependable Secure Comput.*, to be published, doi: 10.1109/TDSC.2021.3093711.

[9] W. Jeong, J. Yoon, E. Yang, and S. J. Hwang, "Federated semi-supervised learning with inter-client consistency & disjoint learning," in *Proc. 9th Int. Conf. Learn. Representations*, 2021, pp. 1–11.

[10] R. Gálvez, V. Moonsamy, and C. Diaz, "Less is more: A privacy-respecting android malware classifier using federated learning," in *Proc. Privacy Enhancing Technol.*, vol. 2021, no. 4, pp. 96–116, 2021.

[11] Y. Tang *et al.*, "Visual and semantic knowledge transfer for large scale semi-supervised object detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 12, pp. 3045–3058, Dec. 2018.

[12] Y. Zhang *et al.*, "Deep matrix factorization with knowledge transfer for lifelong clustering and semi-supervised clustering," *Inf. Sci.*, vol. 570, pp. 795–814, 2021.

[13] T. Wuchner, A. Cislak, M. Ochoa, and A. Pretschner, "Leveraging compression-based graph mining for behavior-based malware detection," *IEEE Trans. Dependable Secure Comput.*, vol. 16, no. 1, pp. 99–112, Jan./Feb. 2019.

[14] A. Abusnaina *et al.*, "A deep learning-based fine-grained hierarchical learning approach for robust malware classification," *IEEE Trans. Dependable Secure Comput.*, to be published, doi: 10.1109/TDSC.2021.3097296.

[15] M. Fan *et al.*, "Android malware familial classification and representative sample selection via frequent subgraph analysis," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 8, pp. 1890–1905, Aug. 2018.

[16] Z. Xu, K. Ren, and F. Song, "Android malware family classification and characterization using CFG and DFG," in *Proc. Int. Symp. Theor. Aspects Softw. Eng.*, 2019, pp. 49–56.

[17] S. Cesare, Y. Xiang, and W. Zhou, "Control flow-based malware variant detection," *IEEE Trans. Dependable Secure Comput.*, vol. 11, no. 4, pp. 307–317, Jul./Aug. 2014.

[18] H. Nguyen, Q. Ngo, and V. Le, "A novel graph-based approach for IoT botnet detection," *Int. J. Inf. Secur.*, vol. 19, no. 5, pp. 567–577, 2020.

[19] A. Azmoodeh, A. Dehghantanha, and K. R. Choo, "Robust malware detection for Internet of (Battlefield) things devices using deep eigenspace learning," *IEEE Trans. Sustain. Comput.*, vol. 4, no. 1, pp. 88–95, Jan./Feb. 2019.

[20] R.-H. Hsu *et al.*, "A privacy-preserving federated learning system for android malware detection based on edge computing," in *Proc. 15th Asia Joint Conf. Inf. Secur.*, 2020, pp. 128–136.

[21] T. D. Nguyen, S. Marchal, M. Miettinen, H. Fereidooni, N. Asokan, and A.-R. Sadeghi, "DÏoT: A federated self-learning anomaly detection system for IoT," in *Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst.*, 2019, pp. 756–767.

[22] M. Cai, Y. Jiang, C. Gao, H. Li, and W. Yuan, "Learning features from enhanced function call graphs for android malware detection," *Neurocomputing*, vol. 423, pp. 301–307, 2021.

[23] H. Zhang *et al.*, "ScanMe mobile: A cloud-based android malware analysis service," *ACM SIGAPP Appl. Comput. Rev.*, vol. 16, no. 1, pp. 36–49, 2016.

[24] R. Feng, S. Chen, X. Xie, G. Meng, S.-W. Lin, and Y. Liu, "A performance-sensitive malware detection system using deep learning on mobile devices," *IEEE Trans. Inf. Forensics Security*, vol. 16, pp. 1563–1578, 2021.

[25] Z. Li, J. Kang, R. Yu, D. Ye, Q. Deng, and Y. Zhang, "Consortium blockchain for secure energy trading in industrial Internet of Things," *IEEE Trans. Ind. Informat.*, vol. 14, no. 8, pp. 3690–3700, Aug. 2018.

[26] S. Luo, X. Chen, Q. Wu, Z. Zhou, and S. Yu, "HFEL: Joint edge association and resource allocation for cost-efficient hierarchical federated edge learning," *IEEE Trans. Wireless Commun.*, vol. 19, no. 10, pp. 6535–6548, Oct. 2020.

[27] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2017, pp. 1273–1282.

[28] N. Papernot, M. Abadi, I. Erlingsson, I. J. Goodfellow, and K. Talwar, "Semi-supervised knowledge transfer for deep learning from private training data," in *Proc. 5th Int. Conf. Learn. Representations*, 2016, pp. 1–12.

[29] Q. Zhang, J. Ma, J. Lou, L. Xiong, and X. Jiang, "Towards training robust private aggregation of teacher ensembles under noisy labels," in *Proc. IEEE Int. Conf. Big Data*, 2020, pp. 1103–1110.

[30] H. Liu, J. Li, and D. Gu, "Understanding the security of app-in-the-middle IoT," *Comput. Secur.*, vol. 97, 2020, Art. no. 102000.

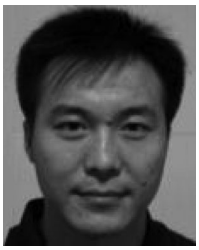
[31] W. Zhou *et al.*, "Discovering and understanding the security hazards in the interactions between IoT devices, mobile apps, and clouds on smart home platforms," in *Proc. 28th USENIX Secur. Symp.*, 2019, pp. 1133–1150.

[32] P. Sivakumaran and J. Blasco, "A study of the feasibility of co-located app attacks against BLE and a large-scale analysis of the current application-layer security landscape," in *Proc. 28th USENIX Secur. Symp.*, 2019, pp. 1–18.

[33] G. Ho, D. Leung, P. Mishra, A. Hosseini, D. Song, and D. A. Wagner, "Smart locks: Lessons for securing commodity Internet of Things devices," in *Proc. 11th ACM Asia Conf. Comput. Commun. Secur.*, 2016, pp. 461–472.

[34] K. Tian, D. Yao, B. G. Ryder, G. Tan, and G. Peng, "Detection of repackaged android malware with code-heterogeneity features," *IEEE Trans. Dependable Secure Comput.*, vol. 17, no. 1, pp. 64–77, Jan./Feb. 2020.
[35] F. Wei, Y. Li, S. Roy, X. Ou, and W. Zhou, "Deep ground truth analysis of current android malware," in *Proc. Int. Conf. Detection Intrusions Malware Vulnerability Assessment*, 2017, pp. 252–276.
[36] S. Mahdavifar, A. F. A. Kadir, R. Fatemi, D. Alhadidi, and A. A. Ghorbani, "Dynamic android malware category classification using semi-supervised deep learning," in *Proc. IEEE Int. Conf Dependable Autonomic Secure Comput.*, 2020, pp. 515–522.
[37] K. Allix, T. F. Bissyande, J. Klein, and Y. L. Traon, "AndroZoo: Collecting millions of android apps for the research community," in *Proc. 13th Int. Conf. Mining Softw. Repositories*, 2016, pp. 468–471.
[38] Virusshare, 2021. [Online]. Available: http://virusshare.com/
[39] T. Kim, B. Kang, M. Rho, S. Sezer, and E. G. Im, "A multimodal deep learning method for android malware detection using various features," *IEEE Trans. Inf. Forensics Secur.*, vol. 14, no. 3, pp. 773–788, Mar. 2019.
[40] W. Wang, M. Zhu, X. Zeng, X. Ye, and Y. Sheng, "Malware traffic classification using convolutional neural network for representation learning," in *Proc. Int. Conf. Inf. Netw.*, 2017, pp. 712–717.
[41] B. Athiwaratkun and J. W. Stokes, "Malware classification with LSTM and GRU language models and a character-level CNN," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, 2017, pp. 2482–2486.
[42] A. N. Jahromi, S. Hashemi, A. Dehghantanha, R. M. Parizi, and K.-K. R. Choo, "An enhanced stacked LSTM method with no random initialization for malware threat hunting in safety and time-critical systems," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 4, no. 5, pp. 630–640, Oct. 2020.
[43] Y. Qin, N. Frosst, S. Sabour, C. Raffel, G. Cottrell, and G. Hinton, "Detecting and diagnosing adversarial images with class-conditional capsule reconstructions," in *Proc. 8th Int. Conf. Learn. Representations*, 2020, pp. 1–12.
[44] M. Xia, L. Gong, Y. Lyu, Z. Qi, and X. Liu, "Effective real-time android application auditing," in *Proc. IEEE Symp. Secur. Privacy*, 2015, pp. 899–914.
[45] L. Onwuzurike, M. Almeida, E. Mariconti, J. Blackburn, G. Stringhini, and E. D. Cristofaro, "A family of droids-android malware detection via behavioral modeling: Static vs dynamic analysis," in *Proc. 16th Annu. Conf. Privacy Secur. Trust*, 2018, pp. 1–10.

Xinjun Pei is currently working toward the PhD degree in the School of Computer Science and Engineering, Central South University, Changsha, China. Since 2017, he has been engaged in the direction of information security. His research interests include deep learning, edge computing, and IoT security.



Xiaoheng Deng (Member, IEEE) received the PhD degrees in computer science from Central South University, Changsha, Hunan, P.R. China, in 2005. Since 2006, he has been an Associate Professor and then a full professor with the School of Computer Science and Engineering, Central South University. He is the chair of RS Changsha Chapter, a senior member of CCF, a member of CCF Pervasive Computing Council, a member of the ACM. He has been a chair of CCF YOCSEF CHANG SHA from 2009 to 2010. His research interests include wireless communications and networking, edge computing, congestion control for wired/wireless network, cross layer route design for wireless mesh network and ad hoc network, social network analysis, distributed computing system.

Shengwei Tian received the BS, MS, and PhD degrees from the School of Information Science and Engineering, Xinjiang University, Urumqi, China, in 1997, 2004, and 2010, respectively. Since 2002, he has been a teacher with the School of Software, Xinjiang University, where he is currently a professor. His research interests include artificial intelligence, natural language processing, and cyberspace security.

Lan Zhang (Member, IEEE) received the BEng and MS degrees in telecommunication engineering from the University of Electronic Science and Technology of China, Chengdu, China, in 2013 and 2016, and the PhD degree in electrical and computer engineering from the University of Florida, Gainesville, Florida, in 2020. She is currently an Assistant Professor with the Department of Electrical and Computer Engineering, Michigan Technological University. Her research interests include wireless communications, vehicular systems, Big Data analysis, and security and privacy issues for various cyber-physical system applications.

Kaiping Xue (Senior Member, IEEE) received the bachelor's degree from the Department of Information Security, University of Science and Technology of China (USTC), in 2003, and the PhD degree from the Department of Electronic Engineering and Information Science (EEIS), USTC, in 2007. From May 2012 to May 2013, he was a postdoctoral researcher with the Department of Electrical and Computer Engineering, University of Florida. Currently, he is a professor with the School of Cyber Security, USTC. His research interests include next-generation Internet architecture design, transmission optimization, and network security. He serves on the editorial board of several journals, including the *IEEE Transactions on Dependable and Secure Computing* (TDSC), *IEEE Transactions on Wireless Communications* (TWC), and *IEEE Transactions on Network and Service Management* (TNSM). He has also served as a (lead) guest editor for many reputed journals/magazines, including the *IEEE Journal on Selected Areas in Communications* (JSAC), *IEEE Communications Magazine*, and *IEEE Network*. He is an IET fellow.