MDHE: A Malware Detection System Based on Trust Hybrid User-Edge Evaluation in IoT Network

Xiaoheng Deng, *Senior Member, IEEE*, Haowen Tang, Xinjun Pei, *Student Member, IEEE*, Deng Li, *Member, IEEE*, and Kaiping Xue, *Senior Member, IEEE*

***Abstract*— With the coming of the Internet of Things (IoT) era, malware attacks targeting IoT networks have posed serious threats to users. Recently, the emerging of edge computing have paved the way for new data processing paradigms in IoT networks, but it is still a challenge for deploying malware detection systems on the IoT devices. This paper develops an IoT malware detection system based on trust hybrid user-edge evaluation, namely MDHE. This system decomposes a large and complex deep learning model into two parts, which are deployed on edge servers and end devices, respectively. Specifically, a trust evaluation mechanism is used to select the trusted devices to participate the model training. Moreover, we develop a private feature generation that leverages a graph mining technology to extract the subgraph features, which then are perturbed by leveraging the differential privacy technology to prevent user privacy from leaking. Finally, we reconstruct the perturbed features on edge server, and propose a Capsule Network (CapsNet) to identify malware. Experimental results show that MDHE can effectively detect malware. Specifically, it can reduce sensitive inference while maintaining the utility of data.**

***Index Terms*— Edge computing, malware detection, trust evaluation, capsule network.**

I. Introduction

WITH the continuous development of IoT, a vast amount of smart IoT devices benefited from the Android platforms [1]. While IoT devices provide users with convenient services, cybercriminals leverage the vulnerabilities of IoT devices to launch malware attacks in IoT networks. Many third-party IoT applications are designed for specific IoT networks, such as healthcare, monitoring, and autonomous driving systems. Third-party vendors deploy these applications on IoT devices with minimal security verification. When the malware intrudes into IoT networks, personal computers, corporate hosts, enterprise or national servers, the private information stored by individuals and the company's trade secrets will be completely leaked, which brings huge economic losses [2], [3]. This motivates us to develop more effective techniques for malware detection.

Machine learning is an analysis and decision-making technology with strong self-learning and adaptive capabilities, which has been widely used in malware detection, showing superior detection performance. However, directly deploying malware detection systems to IoT devices may be difficult in practice. Becuase some IoT devices have limited computing power. Therefore, it is very important to evaluate the computing capability of a IoT device before it participates in model training [4], [5]. Moreover, a promising solution is to implement the parallel computation of neural network through the segmentation of neural network, which can reduce IoT device computing overhead. Specifically, some IoT devices may violate mutually-agreed norms, leading to disruptions in the training and deployment of malware detection algorithms. There is a need for trust management to evaluate the trustworthiness of devices through trust and reputation scores. In this case, the trust management mechanism evaluates the reliability of the input of the IoT device and timely detects device anomalies. In this paper, we propose a trust-evaluation-based hybrid user-edge architecture to ensure the reliability of participating training IoT devices and use a neural network segmentation method to reduce the computing overhead of IoT devices.

Some existing malware detection systems have been explored in the context of user devices [6]. However, it is difficult to train models on those resource-constrained and battery-powered devices. Traditional centralized malware detection systems use the cloud for model training and testing, which makes the training and retraining of the detection model very challenging. Moreover, the back and forth communication also brings other problems, such as the high communication overhead and the data transmission delay, especially when dealing with geographically distributed IoT devices that produce a large amount of data [2], [7], [8]. Therefore, there is

Manuscript received 20 January 2023; revised 11 July 2023; accepted 24 August 2023. Date of publication 25 September 2023; date of current version 13 November 2023. This work was supported in part by the National Natural Science Foundation of China under Project 62172441 and Project 61772553, in part by the National Natural Science Foundation of Hunan Province under Grant 2023JJ30696, in part by the Local Science and Technology Developing Foundation Guided by Central Government through the Free Exploration Project 2021Szvup166, in part by the Key Project of Shenzhen City Special Fund for Fundamental Research under Grant JCYJ20220818103200002, and in part by the Opening Project of State Key Laboratory of Nickel and Cobalt Resources Comprehensive Utilization under Grant GZSYS-KY-2022-018 and Grant GZSYS-KY-2022-024. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. George Loukas. *(Corresponding author: Xinjun Pei.)*

Xiaoheng Deng, Haowen Tang, Xinjun Pei, and Deng Li are with the School of Electronic Information, Central South University, Changsha 410083, China, and also with the Shenzhen Research Institute, Central South University, Shenzhen 518000, China (e-mail: dxh@csu.edu.cn; 928571437@qq.com; pei_xinjun@163.com; d.li@csu.edu.cn).

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).

Digital Object Identifier 10.1109/TIFS.2023.3318947

1556-6021 © 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: University of Science & Technology of China. Downloaded on January 27,2024 at 12:38:38 UTC from IEEE Xplore. Restrictions apply.

TABLE I
COMPARISON BETWEEN MALWARE DETECTION METHODS

Method	Category	Detection location	Feature characterization	Complexity of feature extraction	Learning model	Detection performance	Privacy protection
Sun *et al.* [12]	Static analysis	-	System call sequences	Moderate	DEEPMALWAR	Moderate	No
Chai et. al. [13]	Static analysis	-	Malware image features	High	DPNSA	Low	No
Xu *et al.* [14]	Static analysis	-	API cluster	Moderate	SDAC	Moderate	No
Zhu *et al.* [15]	Static analysis	-	Permission and APIs	High	SHLMD	Low	No
Jeon *et al.* [16]	Static analysis	-	Opcode and APIs	High	HyMalD	Low	No
Tian *et al.* [17]	Dynamic analysis	Cloud	Dynamic features	High	Multi-CNN	Moderate	No
Mishra *et al.* [18]	Dynamic analysis	Cloud	System calls	High	RF	High	No
Vahedi *et al.* [19]	Dynamic analysis	Cloud	Behavioral entropy	High	HMM	Low	No
Our work	Static analysis	Edge	Malicious subgraph	Moderate	CapsNet	High	Yes

a great need for decentralized detection solutions. In addition, there are some security issues related to the learning model, such as unauthorized access, lack of control or availability, privacy risks, etc. Therefore, we propose extending the malware detection system to edge computing paradigm, where computing resources are pushed to edge servers that are closed to user terminal devices. Edge computing meets the need for low latency [9] compared to cloud computing. Finally, security providers can protect users from malware infections.

Although machine learning is practical and essential for malware detection, traditional shallow machine learning models (*e.g.* Linear Discriminant Analysis (LDA)) have limited learning capabilities in the modeling complex malware behavior patterns, resulting in low detection accuracy. Centralized collection of Android Package (APK) for high-performance model training may greatly increase the communication overhead. How to utilize the large number of iot applications to build more effective and accurate detection models remains a challenge. More importantly, the private information of an app (*e.g.* Application Programming Interface (API) calls [10] and permission configurations [11]) could be revealed during the data upload process, which could be a potential threat. The adversary can use the exposed private information to infer users' interests to target advertising. Also, based on such sensitive information, the adversary can even generate adversarial samples to avoid detection. Therefore, it is important to protect user privacy from threats.

In this paper, we develop a trust hybrid user-edge evaluation based IoT malware detection system (MDHE), which can effectively model the malicious behaviors. In this system, user device and edge server collaboratively train a complex neural network, without divulging specific private information. Specifically, a Trust Evaluation (TE) mechanism is proposed to select trusted devices to participate in our model training. Each user device uses a feature generation module to extract useful information from a larger number of applications as the intermediate features. Then, a private attention module is performed to perturb the extracted features by leveraging the differential privacy technology, which can prevent adversaries from inferring sensitive information. After that, MDHE uploads the perturbed features to the edge server for training and fine-tuning a deep learning model with powerful detection capabilities. Our major contributions are as follows.

- We propose a trust-evaluation-based hybrid user-edge architecture to reduce the communication overhead and delay, where user devices and edge servers collaboratively build a malware detection model.
- We integrate a feature generation module and a private attention module into MDHE to effectively model the malicious behaviors, while protecting users from malware infections.
- Extensive experiments on real-world sample data demonstrate MDHE's effectiveness for IoT malware detection. The results indicate that MDHE can improve the detection performance and provide sufficient privacy and robustness guarantee.

The rest of this paper is organized as follows. Firstly, we briefly describe the related work in Section II, and present a thorough overview of our MDHE in Section IV. Then, we evaluate the proposed MDHE model in Section V. Moreover, we apply MDHE to engineering applications in Section VI. Finally, this paper summarizes in Section VII.

II. Related Work

A. Deep Learning-Based Malware Detection

To mitigate malware threats in the IoT networks, many efforts have been devoted to develop DL-based malware detection systems. Table I lists some representative malware detection methods. These systems can analyze longer sequences of system calls and achieve more accurate classification by extracting higher levels of information. For example, Sun et al. [12] introduced a malware detection framework called PROPEDEUTICA, which incorporates a novel deep learning architecture (DEEPMALWARE). This architecture utilizes multistream inputs to combine the strengths of machine learning (ML) and deep learning (DL) for efficient and effective real-time malware detection. Chai et. al. [13] formulated unknown malware detection as a Few-Shot Learning problem. They presented DPNSA, a malware detection framework specifically designed for few-shot malware detection. DPNSA incorporates a dual-sample dynamic activation function to reduce the impact of irrelevant features. To tackle the issue of model aging, Xu et al. [14] proposed a novel slow-aging solution named SDAC, which clusters all APIs based on their semantic distances. Zhu et al. [15] proposed SHLMD, a hybrid deep network representation learning method that captures the relevant features in different levels of granularity

Authorized licensed use limited to: University of Science & Technology of China. Downloaded on January 27,2024 at 12:38:38 UTC from IEEE Xplore. Restrictions apply.

information. Similarly, Jeon et al. [16] proposed HyMalD, a hybrid malware detection scheme that combines bidirectional Long short-term memory (Bi-LSTM) and spatial Pyramid Pool Network (SPP-Net) for IoT malware detection. However, existing DP-based malware detectors have difficulties in runtime detection due to the performance overhead [1], [9]. While some existing malware detection systems have been explored in the context of user devices [6], state-of-the-art DL models, which consist of a large number of parameters and deep layers, often suffer from slow computations on the resource-constrained and battery-powered devices. Due to the high computational requirements of DL models, deploying these DL-based malware detection systems on such devices becomes impractical. In this paper, we propose a trust-evaluation-based hybrid user-edge architecture that decomposes a large and complex deep learning model into two parts, i.e., the edge-server part and the end-device part. This design alleviates the need for computing resources on user devices.

B. IoT Malware Detection

In IoT networks, IoT devices generate large amounts of data, which are typically processed locally due to limited transmission capacity. Distributed computing architecture has the advantages of system scalability, short response time, data security and privacy. Teerapittayanon et al. [4] used a distributed deep learning model to implement cloud-edge-end collaborative inference through the segmentation of neural network. Similarly, Kang et al. [5] investigated neural network segmentation to address the requirements of low latency and energy consumption. These studies achieve parallel computation by segmenting the neural network. However, this leads to the transmission of a large number of intermediate features, greatly increasing the communication overhead. In this paper, we divide the neural network into two parts and deploy them to the edge server and the IoT device, respectively. This approach reduces communication overhead by reducing the amount of data to be transmitted.

Moreover, cloud-based malware detection methods have emerged as a promising solution to enhance the security of IoT devices. To address the increasing malware threats in cloud environments, Tian et al. [17] proposed a dynamic analysis-based malware detection method that utilizes a lightweight agent to collect runtime utilization and memory object information from the target virtual machine (VM), and employs a multi-CNN model for efficient malware detection. This method can effectively detect malware without any modification to the guest VM or hypervisor. Mishra et al. [18] proposed an introspection based security approach to mitigate malware attacks in the cloud. They analyzed the run-time behavior of processes, and extracted the n-gram features. Vahedi et al. [19] proposed a cloud-based malware detection framework, which monitors the behavioral characteristics of files within a sandbox environment, and classifies them into malware families based on behavioral patterns. While these methods utilize cloud infrastructure's computational power and storage capabilities for malware analysis and detection, they might not satisfy the real-time needs of IoT users in terms of timely detection and response. In this paper, we propose extending the malware detection system to the edge computing paradigm, which is closed to user terminal devices, aiming to alleviate the computational burden on IoT devices.

C. Trust Evaluation

Trust evaluation is widely used in peer-to-peer computing and distributed cloud computing, such as medical networks and automobile networks. In social media health networks, Tang et al. [20] proposed a personalized healthcare solution that provides trusted and privacy-preserving services. This increases the level of trust between caregivers and patients by authentic ratings. Moreover, in [21], Tang et al. proposed a privacy-preserving fog-assisted scheme PFHDS to share health data. Then, they designed an enhanced attribute-based encryption method to effectively deliver health care. Blaze et al. [22] established a trust management system. This paper focuses on how to improve the security and efficiency of interaction between user devices and edge server, and how to filter trusted devices.

D. Privacy Protection

Although edge computing has some advantages in data analysis, considering the unreliability of edge server, it faces the risk of privacy disclosure. A recent regulation of European Union stipulates that companies should carefully collect and use personal data. Adversaries can use sensitive data on edge servers to infer users' private information [20], [21]. For example, a user uploads the features of a given app to an edge server. If adversaries gain access to these features, they can infer some private information (*e.g.*, user's interests and preferences). There is a fundamental conflict between protecting the private information and retaining the utility of the data [23], [24]. Some studies have used encryption to address this limitation. However, the encryption-based methods are difficult to deploy on user devices due to the limited resources and high computing costs [2], [7]. Although many existing machine learning-based malware detections have achieved good performance, few efforts have considered protecting user privacy information. In this paper, we apply differential privacy to effectively protect user privacy.

III. System Model

We propose a hybrid architecture MDHE based on edge computing for malware detection, which includes an edge layer and an IoT device layer. Each edge server is connected to a set of IoT devices covering an area. Raw data is transferred to the edge layer, which is composed of many high-performance computing nodes or edge servers. This paper embeds a pre-trained malware detection model (see Section IV-C for details) into our hybrid architecture MDHE. We choose the middle layer of model as the pivot and divide the entire MDHE into two main parts. 1) The first part contains feature generation module and private attention module. User device and edge server cooperate to detect malware. Firstly, feature generation module and private attention module are pre-trained and deployed on user device, where the feature generation module performs minimal processing on the raw data to extract

Authorized licensed use limited to: University of Science & Technology of China. Downloaded on January 27,2024 at 12:38:38 UTC from IEEE Xplore. Restrictions apply.

features, and the private attention module uses the attention mechanism to extract key features. Specifically, we use a private intermediate layer to inject random noise into the features. Then, user device uploads the intermediate features to edge server for further processing. Based these features, the malware classifier on edge server returns the expected results to user device. 2) The second part uses a CapsNet to build a robust and accurate malware classifier. By leveraging the computational resources and storage capacity of the edge server, we can optimize the performance and effectiveness of the malware classifier, enabling it to accurately identify and classify malware.

A. Threat Model

IoT devices lacking computing resources are facing serious security threats. We list some of the threats of deploying a malware detection system in IoT network.

- An adversary can gain access to a user's private data through malware attacks, such as using malware to steal user data or executing a distributed denial of service botnet. Moreover, the Android system provide some coarse-grained permissions. Thus, the malware can access the external device and create some rules to guide cloud server to interact with the sensing devices.
- As mentioned earlier, the extracted API calls and permission features often implicitly reflect user habits. The adversary could use this information to infer the gender and interest of users for targeted advertising [23], and even generate adversarial samples to avoid detection [24]. It is a challenge to design an effective feature generation module that retain useful information while protecting sensitive information from adversaries.
- When a user performs malware detection on a local device, the device is usually unable to cope with the resource-consuming malware detection system due to limited resources and data processing capabilities. In addition, traditional cloud-based centralized detection systems may lead to a lot of communication overhead.
- Due to the limited computing power of IoT devices, the overhead of feature extraction must be minimized. However, discarding information may also have a negative impact on malware detection. This requires careful design of feature reduction schemes.

B. Design Goal

IoT networks are now facing more targeted malware attacks. To prevent malware attackers from attacking IoT infrastructure, we propose a trust-evaluation-based hybrid user-edge architecture. Based on the identity, performance, and behavior of user devices, the trust evaluation can be used to select trusted devices to participate in model training, thereby providing coarse-grained protection for user privacy. To reduce communication costs and minimize latency in data transmission, this paper designs a layer separation mechanism, where end devices and edge servers cooperate to build a malware detection model. On the end-device side, a private attention module is used to extract key features by considering the probability weight distribution. However, these features remain relatively sensitive, and could be exploited by adversaries to infer sensitive user information. Consequently, we use the DP mechanism to perturb the features and upload them to the edge server. Because of the limited computing power of end devices, we extend a deep learning model to the edge. On the edge-server side, we develop a malware detection model based CapsNet. By utilizing the powerful computing power of edge servers, we are able to effectively train the CapsNet model on a large malware dataset.

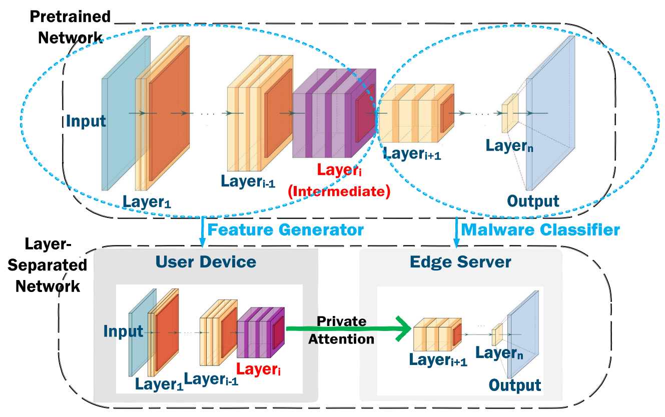


Fig. 1. The proposed hybrid user-edge architecture.

IV. Our Proposed MDHE

A. Overview

This paper proposes a trusted hybrid user edge evaluation based malware detection system, which preserves data utility while reducing sensitive inference and system overhead. Fig. 1 shows the architecture of MDHE. User device and edge server collaboratively train a complex neural network using user data, without divulging specific private information. In our system, a TE mechanism is used to select trusted devices to implement the model training. Then, each user device uses a feature generation module to extract useful information from apps as the intermediate features. Next, a private attention module leverages the differential privacy (DP) technology to perturb the features. Finally, user devices upload the perturbed features to the edge server for training and fine-tuning a deep learning model.

B. Trust Evaluation

We use edge server with strong computing and storage capabilities to provide TE for user devices to ensure the reliability of participating IoT devices and avoid potential attacks. Eventually, a safe, reliable and efficient malware detection model is implemented.

Before model training, the edge server screens out trusted IoT devices. When a device participates, it communicates with the nearest edge server, which then checks to see if the device exists in the current list of trusted devices e.g., existing historical interaction. After that, the server authorizes this device to participate in model training. Otherwise, the

Authorized licensed use limited to: University of Science & Technology of China. Downloaded on January 27,2024 at 12:38:38 UTC from IEEE Xplore. Restrictions apply.

server evaluates this device, which includes three evaluation methods, i.e., Identity Evaluation, Capability Evaluation, and Behavior Evaluation. Finally, the server adds this device to the list of trusted devices and authorizes it to participate in training. These three evaluations are described in detail below.

1) **Identity Evaluation** (*IE*): User devices are highly mobile and dynamic, which makes their identity authentication difficult. Therefore, each user device performs real-name registration to ensure the authenticity and uniqueness of identity in the edge network. Specifically, devices that are added to the system for the first time are given a unique ID number, and then the system saves evaluation information. Before the device with unique ID number participates in the model training, the edge server authenticates the user device, which verifies whether it match the corresponding ID number. If the device obtains the authentication, *IE* is set to 1. Otherwise, the device is refused to participate in the model training.
2) **Capability Evaluation** (*CE*): The computing power of user devices is usually limited. Before the user device participates in the training, it submits the device performance configuration information to the edge server, and then it determines whether the device capability meets the requirements of the model training. In this paper, we use four representative attribute parameters to evaluate the user devices, including CPU, Disk, Memory and Online-time. If all four performance attributes meet the edge server's performance requirements, *CE*=1, otherwise, the edge server will reject the user device.
3) **Behavior Evaluation** (*BE*): The *BE* can be measured by all historical interactions between edge servers and IoT devices. A higher value of *BE* means less malicious behavior for the user device to interact with the edge server. Conversely, a lower value for *BE* means that the user device is more likely to be considered a malicious device. Note *BE* ∈ [0,1]. When *BE* is below the preset threshold, the device will be refused to participate in the model training.

According to the current behavior and historical behavior, we update the value of *BE* of the target user device by weighting *BE* from two different angles obtained by interaction. This is because different edge servers may observe behavior and evaluate target user devices based on different conditions. With proper weighting, highly accurate values of *BE* can be obtained. The value of *BE* is synthesized by a direct evaluation value (DV) and an indirect evaluation value (IV). DV is the trust value generated by the direct interaction between edge server and user device. Based on satisfaction of the returned results, the edge server can give a value of DV for this user device, which can be seen as a direct interactive experience. Moreover, IV is calculated by collecting the evaluation value of the edge servers that have historically interacted with the user device, which can be seen as an indirect experience for edge servers. Fig. 2 shows the workflow of TE. The main steps are as follows:

1) When the network system is initialized, for each user device, the value of *BE* is set to 0.3.

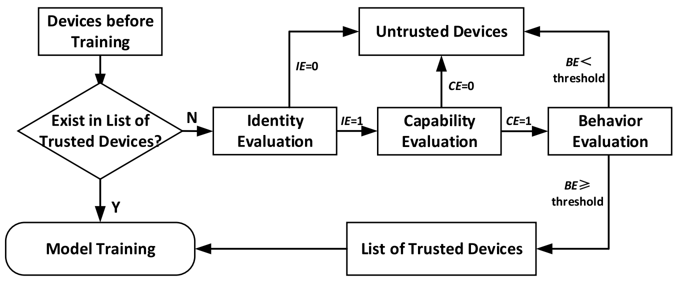


Fig. 2. The overview of trust evaluation.

2) For a new user device D, based the value of *IE*, the edge server adds it to the list of trusted devices list, and records its performance attribute parameters.
3) After the edge server S enters the network, it declares the performance parameters required for model training, and accordingly filters out user device D that meets the requirements.
4) Before the edge server S interacts with the user device D, the edge server S queries the value of *BE* of the user device D, and decides whether to add user device D. We set the threshold for *BE* to 0.2.
5) After the model training is finished, the edge server S evaluates the user device D, and then uses its evaluation value as the direct evaluation value to update the value of *BE*.

In our proposed TE, based on the *BE* of the user device, the edge server S adjusts the priority of the user device D. Then, the edge server S give a direct evaluation value DV_{SD} and an indirect evaluation value IV_{SD} for the user device D. Finally, the edge server S updates the value of *BE* of the user device D.

$$BE = \alpha DV_{SD} + \beta IV_{SD}, \tag{1}$$

where α and β are two tunable parameters, satisfying $\alpha + \beta = 1$, When $\beta > \alpha$, the server S pays more attention to historical interactions. On the contrary, when $\beta < \alpha$, the rater server S pays more attention to current interactions between the server and devices. This means that the value of *BE* is calculated based on current and historical behavior.

The edge server S can directly give the direct evaluation value $DV_{SD}(0 < DV_{SD} \leq 1)$ after the user device D participates in the model training. The indirect evaluation value is the evaluation value given by the edge server that has a history of interaction with the user device D. IV_{SD} can be calculated by:

$$IV_{SD} = \frac{1}{n} * \sum_{i=1}^{n} T(t_i) DV_{iD}, \tag{2}$$

where $T(t_i) = e^{-\delta(t-t_i)}$. n is the number of edge servers that have a history of interacting with user device D, and DV_{iD} indicates the direct evaluation value of i-th edge server. Here, $T(t_i)(1 \leq i \leq n)$ is a time decay factor that measures the freshness of the evaluation. We use a predefined parameter δ to control the effect of the time decay factor $T(t_i)$. t indicates the current time. In historical interactions, the latest evaluation value is given a higher weight.

Authorized licensed use limited to: University of Science & Technology of China. Downloaded on January 27,2024 at 12:38:38 UTC from IEEE Xplore. Restrictions apply.

C. Malware Detection

In this section, we use the Malicious Subgraphs (MSGs) and permissions as features. Our system consists of three stages: feature generation, private attention, and malware classification. In feature generation stage, we extract the MSGs and permissions from APK and then merge them into feature vectors. In private attention stage, we extract important features and inject noise into them. Finally, in malware inference stage, we use a malware classifier to detect malware.

1) Feature Generation Module on User Device: In order to characterize the program semantics of an application, we use a disassembly tool (such as ApkTool) to get the Dalvik code from the APK files. Android malware usually calls some sensitive APIs and permissions for manipulating sensitive data. We use the Pscout tool to get a set of sensitive APIs $SAS = \{v_1, v_2, \ldots, v_{8910}, v_{8911}\}$ and a set of permissions $\{p_1, p_2, \ldots, p_{455}\}$. By doing this, a total of 8911 sensitive APIs and 455 permissions can be available.

To distinguish the importance of different sensitive APIs, each sensitive API is assigned an importance weight. Based on the TF-IDF, we make the maliciousness degree of sensitive API v_k be in positive correlation with the percentage of malware that invoke the sensitive API v_k and in negative correlation with the percentage of all apps that invoke the sensitive API v_k. In our case, the maliciousness degree $md(v_k)$ of a sensitive API v_k is calculated by:

$$md(v_k) = P_m(v_k) * log\frac{Num_m + Num_b}{N_m(v_k) + N_b(v_k)}, \quad (3)$$

where $P_m(v_k)$ denotes the percentage of malware samples that invoke sensitive API v_k. Num_b and Num_m denote the number of benign and malware samples, respectively. $N_b(v_k)$ and $N_m(v_k)$ denote the number of benign and malware samples that invoke the sensitive API v_k, respectively.

Moreover, we use two common tools, Androguard and graph visualization software (Gephi), to create Function Call Graph (FCG), which contains methods (i.e., nodes in FCG) and corresponding call relationships (i.e., edges in FCG) between them. Then, we extract the program semantics and the structural information to describe the malicious behavior of the malware.

After careful analysis of numerous malware samples, we found that benign samples and malware samples have different sensitive API call patterns. The analysis of the whole FCG is very complicated and inefficient because the FCG has thousands of nodes. Typically, malicious behavior is reflected in only a small part of FCG. In order to describe the malicious behavior, we extract sensitive API nodes and their neighboring nodes from FCG to construct the Malicious Subgraphs (MSGs). Algorithm 1 and Fig. 3 shows the generation of a MSG.

In order to build MSG, FCG and SAS are used as inputs to Algorithm 1. For each sensitive API v_k in the set of sensitive APIs SAS, we define its neighbor set to be $\mathcal{N}(v_i)$. Also, we define a distance function $dis(v_k, v_i^k)$, which returns the length of the shortest path. To reduce the size of FCG, we only consider the 1-hop neighbors of each sensitive API node v_k, which are recorded in V_k, as described in Algorithm 1 (Line 2-9). In Algorithm 1 (Line 10-14), we first use a function $RemoveLibNodes(V_k)$ to remove nodes that represent the third-party libraries, and then use a function $RemoveEdge(E_k)$ to keep the edges corresponding to the remaining nodes in V_k. Next, Algorithm 1 (Line 16-22) extracts the MSG feature FV.

Algorithm 1 The generation of $MSGs$

```
Input: $FCG = (V, E)$, $SAS$;
Output: $MSG$;
1: $MSG \leftarrow \emptyset$;
2: for each $v_k \in SAS$ do
3:     $\mathcal{N}(v_i) \leftarrow \{v_1^k, v_i^k, \ldots, v_n^k\}$;
4:     $V_k \leftarrow \emptyset$;
5:     for each $v_i^k \in \mathcal{N}(v_i)$ do
6:         if $dis(v_k, v_i^k) \leq 1$ then
7:             $V_k = V_k \cup \{v_i^k\}$;
8:         end if
9:     end for
10:    $V_k \leftarrow RemoveLibNodes(V_k)$;
11:    $E_k \leftarrow RemainingEdge(E_k)$;
12:    $MSG_k \leftarrow (V_k, E_k)$;
13:    $MSG = MSG \cup \{MSG_k\}$;
14: end for
15: $FV^{(s)} \leftarrow \emptyset$;
16: for each $MSG_k \in MSG$ do
17:    $SF_k \leftarrow \{I(MSG_k) \cdot md(MSG_k)\}$;
18:    $FV^{(s)} = FV^{(s)} \cup \{SF_k\}$;
19: end for
20: $FV^{(p)} = \{P_1, P_2, \ldots, P_m\}$;
21: $FV = \{FV^{(s)}, FV^{(p)}\}$;
22: return $MSG$
```

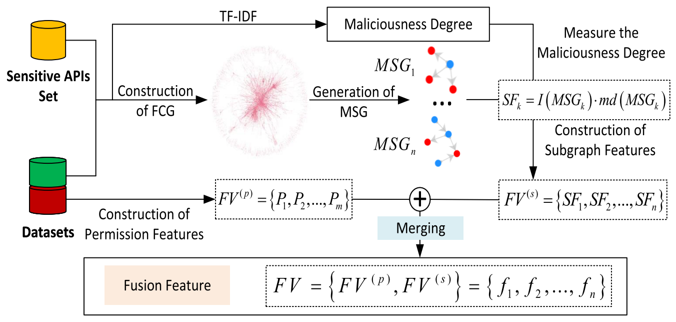

TF-IDF
Maliciousness Degree
Measure the Maliciousness Degree
Sensitive APIs Set
Construction of FCG
Generation of MSG
MSG_1
MSG_n
$SF_k = I(MSG_k) \cdot md(MSG_k)$
Construction of Subgraph Features
Datasets
Construction of Permission Features
$FV^{(p)} = \{P_1, P_2, \ldots, P_m\}$
Merging
$FV^{(s)} = \{SF_1, SF_2, \ldots, SF_n\}$
Fusion Feature
$FV = \{FV^{(p)}, FV^{(s)}\} = \{f_1, f_2, \ldots, f_n\}$

Fig. 3. The generation of MSGs.

Specifically, we measure the maliciousness degree of MSG_k by:

$$md(MSG_k) = \sum_{v_k \in SSA(MSG_k)} md(v_k), \quad (4)$$

where $md(v_k)$ is the maliciousness degree of *k-th* sensitive API nodes. Considering that each MSG's contribution to the detection model should be different, the accuracy of malware detection may be affected if the same weight is assigned to each MSG. To solve this problem, we adaptively add a weight $I(MSG_k)$ to the maliciousness degree of MSG_k, which is

Authorized licensed use limited to: University of Science & Technology of China. Downloaded on January 27,2024 at 12:38:38 UTC from IEEE Xplore. Restrictions apply.

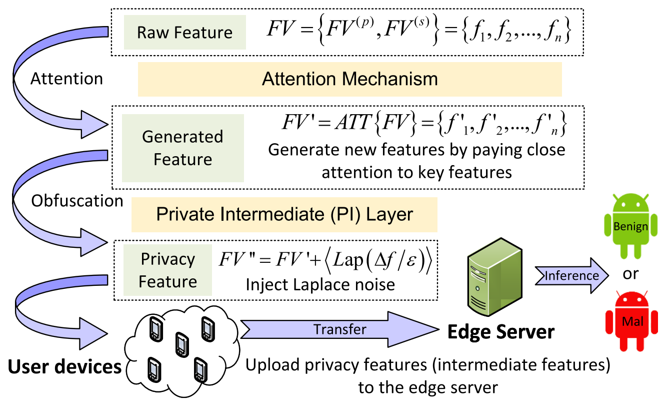

Raw Feature
$FV=\{FV^{(p)},FV^{(s)}\}=\{f_1,f_2,...,f_n\}$
Attention
Attention Mechanism
Generated Feature
$FV'=ATT\{FV\}=\{f'_1,f'_2,...,f'_n\}$
Generate new features by paying close attention to key features
Obfuscation
Private Intermediate (PI) Layer
Privacy Feature
$FV''=FV'+\langle Lap(\Delta f/\varepsilon)\rangle$
Inject Laplace noise
Inference
Benign
or
Mal
Transfer
Edge Server
User devices
Upload privacy features (intermediate features) to the edge server

Fig. 4. The private attention module on user device.

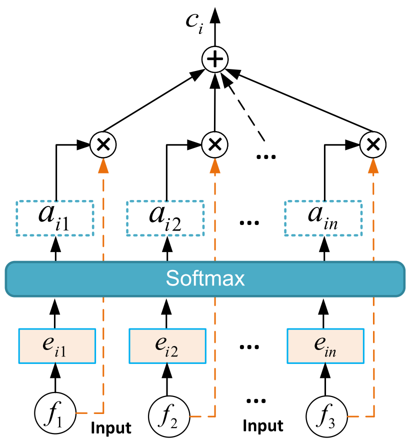


Fig. 5. The attention mechanism used in MDHE.

calculated by:

$$I(MSG_k) = max\{1, \frac{|V_k| * n}{\sum_{j=1}^{n} v_j}\}, \tag{5}$$

where $|V_k|$ denotes the number of nodes in MSG_k, and n denotes the number of all MSGs. Finally, we can obtain the subgraph feature vector $FV^{(s)}$ as follows.

$$FV^{(s)} = \{SF_1, SF_2, \ldots, SF_n\} \tag{6}$$

$$SF_k = \{I(MSG_k) \cdot md(MSG_k)\} \tag{7}$$

where SF_k is the subgraph feature of MSG_k. Next, we extract permission features $FV^{(p)}$ to provide more comprehensive malicious behavior information as follows.

$$FV^{(p)} = \{P_1, P_2, \ldots, P_m\} \tag{8}$$

Specifically, permission features are complementary to the MSG features. Finally, we integrate permission features into MSG feature extraction as follows.

$$FV = \{FV^{(s)}, FV^{(p)}\} \tag{9}$$

After that, FV is used for private attention module on user device.

2) Private Attention Module on User Device: In this part, we introduce the details of attention mechanism and privacy protection on user devices in MDHE. Fig. 4 shows an overview of our private attention module. In the feature generation module, we extract the feature set FV, consisting of the subgraph or permission features, each of which contributes differently to the program semantics. For knowledge distillation, we stacked an attention mechanism on the feature generation module to extract such features that are important to the meaning of the semantic expression.

The attention mechanism can be viewed as a composite function that highlights the impact of key features by calculating the probability distribution of attention. The attention mechanism structure is shown in Fig. 5. We fed FV into the attention mechanism to generate new features, i.e., $FV' = ATT(FV)$. In the attention mechanism, the context vector c_i for the attention allocation coefficient a_{ij} can be calculated as follows:

$$c_i = \sum_{j=1}^{n} \alpha_{ij} FV, \tag{10}$$

Then, we calculate the attention allocation coefficient a_{ij} as follows.

$$\alpha_{ij} = \frac{exp(e_{ij})}{\sum_{k=1}^{n} exp(e_{ik})}, \tag{11}$$

where e_{ij} is an alignment model that scores how well the inputs around position j and the output at position i match, which is calculated by:

$$e_{ij} = a(h_{i-1}, FV), \tag{12}$$

where ann_j denotes the j-th annotation. Finally, the attention mechanism generates new features $FV' = \{f'_1, f'_2, \ldots, f'_n\}$, which enhances the representations of important features. Algorithm 2 (Line 1-2) presents the procedure of the private attention module.

Algorithm 2 Private attention module on the user device

```
Input: Privacy budget $\epsilon$, $PI$ layer
Output: The extracted features $FV$
1: Calculate the importance weight scores $a_{ij}$ using Eqs. (11)-(12) and extract context vector $c_i$ using Eq. 10;
2: Generate new features $FV' = \{f'_1, f'_2, \ldots, f'_n\}$ by paying close attention to key features;
3: Inject random noise into weights of the private intermediate (PI) layer;
4: $FV'' = (f'_1, f'_2, \ldots, f'_n) + \langle Lap(\Delta f/\epsilon)\rangle^d$;
5: return $FV''$.
```

The extracted MSG features can retain useful information that contains a lot of user privacy. These features are collected and sent to the data center for centralized processing. However, the transmission of data introduces privacy concerns, because they are vulnerable to privacy inference attacks, such as member inference attacks, model inversion attacks, etc. It becomes essential to ensure the secure and efficient transmission of this data.

Moreover, we construct a private intermediate (PI) layer which applies Differential Privacy (DP) to protect user privacy, where user devices perturb their feature vectors before transmitting them to the edge server. The DP serves as a rigorous and principled approach to privacy protection, enabling the

Authorized licensed use limited to: University of Science & Technology of China. Downloaded on January 27,2024 at 12:38:38 UTC from IEEE Xplore. Restrictions apply.

analysis and utilization of sensitive data while preserving the privacy of individuals.

As shown in Algorithm 2 (Line 3-5), we use a Laplace Mechanism (LM) to implement ϵ-DP, where the DP noise is obtained from the appropriate Laplacian distribution. Now we give the formal definition of the DP as follows.

Definition 1 (ϵ-Differential Privacy): A randomized algorithm M satisfies ϵ-DP, where $\epsilon \geq 0$, iff for any two neighboring databases $\mathcal{D}$, $\mathcal{D}'$ differing in a single item, and for all possible output set $O \subseteq \text{Range}(M)$, we have:

$$\Pr[\mathcal{M}(\mathcal{D}) \in O] \leq e^{\epsilon} \Pr\left[\mathcal{M}\left(\mathcal{D}'\right) \in O\right]$$

where ϵ represents the privacy budget, which controls the privacy-utility trade-off.

A smaller value of ϵ offers a stronger privacy guarantee, but at the expense of data utility. This implies that the probability of producing a specific outcome from the randomized algorithm M on dataset D should be similar to the probability of producing the same outcome on dataset $\mathcal{D}'$, regardless of the presence or absence of any individual's data. This guarantees the preservation of individuals' privacy. Moreover, we formally define the LM as follows.

Definition 2 (Laplace Mechanism): Let f be a function $f: \mathcal{D} \leftarrow \mathbb{R}^d$, we have:

$$O(\mathcal{D}) = f(\mathcal{D}) + \langle Lap(\Delta f/\epsilon)\rangle^d, \tag{13}$$

where $\Delta f = max_{\mathcal{D},\mathcal{D}'}||f(\mathcal{D}) - f(\mathcal{D}')||$ is the l_1-sensitivity of f, and d is the feature dimension.

For a random training batch, we inject Laplace noise into each feature f_i', which is represented as:

$$\begin{aligned} FV'' &= FV' + \langle Lap(\Delta f/\epsilon)\rangle^d \\ &= (f_1', f_2', \ldots, f_n') + \langle Lap(\Delta f/\epsilon)\rangle^d, \end{aligned} \tag{14}$$

where FV'' denotes the perturbed features, i.e., the intermediate features. Algorithm 2 summarizes the LM of MDHE. Finally, Eq. 14 can be rewritten as

$$FV'' = (f_1'', f_2'', \ldots, f_n'') \tag{15}$$

3) Malware Classifier on Edge Server: After constructing the intermediate features FV'' in the privacy neural layer on user devices, we upload them to an edge server where we train a CapsNet-based classifier for malware detection. Compared to the standard Convolutional Neural Network (CNN), the CapsNet introduces a novel approach by transforming scalar inputs into vectors, facilitating more effective storage of features. CapsNet employs capsules, which are groups of neurons that encapsulate vectorized information, to provide a more expressive and informative method of encoding. The vectorized representation enables capsules to perform complex internal calculations on inputs, and learn how to represent and reconstruct a given sample like an autoencoder.

In our case, the perturbed feature vectors are fed to CapsNet. Then, the filters $W = w_1, \ldots, w_v$ are used to encapsulate the features from the same position across different windows into their corresponding capsules. CapsNet preserves and manipulates complex patterns by capturing the spatial relationships and hierarchical structure of features embedded in vectors.

Algorithm 3 Malware classifier on the edge server

```
Input: Perturbed features $FV'' = (f_1'', f_2'', \ldots, f_n'')$, hidden layers $H$, privacy budget $\epsilon$;
Output: The trained malware detection model;
1: Obtain the perturbed features $FV''$;
2: while not converged do
3:     Reconstruct the perturbed features $FV'' = (f_1'', f_2'', \ldots, f_n'')$ and extract the latent deep information using Eqs. (16)-(20);
4:     Update model parameters;
5: end while
```

Algorithm 3 illustrates the learning process of the CapsNet model. Here, the capsule vector is calculated as follows.

$$u_{ij} = z_i \cdot w_j, \tag{16}$$

$$u_i = g([u_{i1}, u_{i2}, \ldots, u_{iV}]), \tag{17}$$

where z_i denotes the i-th operation result in the previous layer, and g is a non-linear squashing function.

Furthermore, the dynamic routing algorithm is used to ensure the generation of accurate outputs in the form of low-level capsule vectors to higher-level parent capsules within the CapsNet. Through dynamic routing, the CapsNet can encode and encapsulate the inherent spatial relationships that exist between individual parts and their collective representation as a whole. Specifically, a squashing function is used to constrain the length $\parallel p_i \parallel$ in the interval [0, 1].

$$g(x) = squash(x) = \frac{\parallel x \parallel^2}{1 + \parallel x \parallel^2} \frac{x}{\parallel x \parallel}, \tag{18}$$

$$\hat{u}_{j|i} = W u_i, \tag{19}$$

Then, we calculate the capsule v_j by the weighted sum of all $\hat{u}_{j|i}$.

$$v_j = g(\sum_i c_{j|i} \hat{u}_{j|i}), \tag{20}$$

where $c_{j|i}$ denotes the coupling coefficients.

D. Security Analysis

In this paper, we extract $MSGs$ from the FCG which retains only sensitive API nodes with their neighbors, while discarding unimportant information. By doing this, the detection accuracy can be guaranteed, and the feature generation time is reduced. Moreover, we use the trust evaluation mechanism to manage the credibility data of user equipment from three levels of identity evaluation, capability evaluation and behavior evaluation. Then, the edge server selects trusted devices for model training. In summary, the trust evaluation mechanism can ensure that the learning model is not interfered by malicious devices in the training process, while preventing the detection accuracy from being affected.

The attention mechanism gives higher weights to important features, which can improve the detection accuracy. In addition, CapsNet discards the pooling layer used in the CNN to ensure that the required performance can be achieved with less training data. Similarly, CapsNet is a local prediction

Authorized licensed use limited to: University of Science & Technology of China. Downloaded on January 27,2024 at 12:38:38 UTC from IEEE Xplore. Restrictions apply.

TABLE II
MAIN DATASETS USED IN OUR EVALUATION STUDIES

Datasets	Benign apps		Malware		#Total
	Source	#Apps	Source	#Apps	
AndroZoo	Google Play	10,010	AndroZoo	12,121	22,131
Drebin	Google Play	10,010	Drebin	5,560	15,570
MalDroid	Google Play	10,010	MalDroid	8,955	18,965
VirusShare	Google Play	10,010	VirusShare	10,456	20,466

of the whole and can extract important features through the dynamic routing mechanism. Therefore, the effectiveness of the CapsNet can be demonstrated. Finally, we use differential privacy to perturb features to ensure that the users' privacy can not compromised. In our hybrid user-edge architecture, user devices and edge servers can cooperate to detect malware, thereby reducing communication costs, improving detection performance, and protecting user privacy.

V. EVALUATION

In this section, we first introduce datasets and metrics used in our experments. Then, we compare our MDHE with other methods. Afterward, we evaluate the run-time of our method, and discuss the effectiveness of private attention module.

A. Datasets and Metrics

We used four well-known baseline datasets to evaluate the performance of MDHE, including AndroZoo [25], Drebin [26], MalDroid [27] and VirusShare. The dataset description is shown in Table II. The first dataset, AndroZoo [25] dataset, consists of all published applications from 2010 to the present. The labels for this dataset were assigned based on the results given by dozens of different Anti-Virus (AV) scanners. In our experiments, we used 12,121 malware samples. Specifically, all benign samples were collected from the official Google Play app market. For the Drebin dataset, the malware samples were obtained from [26], and were collected from 2010 to 2012. Each sample was labeled as one of 179 malware families, such as Geinimi, FakeDoc, and others. Another dataset, MalDroid [27], is a recent Android malware dataset that encompasses five distinct categories: Banking malware, SMS malware, Riskware, Adware, and Benign. We randomly selected 8,955 malware samples for our experiments. VirusShare is an online platform that hosts a wide range of malware samples. To evaluate the performance of MDHE, we utilized 10,456 malware samples from VirusShare, each of which was analyzed through various AV scanners.

We randomly select 80% samples and 20% samples as the training set and the test set, respectively. All experiments do not involve any re-sampling, which avoids the occurrence of bias or overfitting. Therefore, training and test set do not have any common samples. Finally, we used four datasets to evaluate the effectiveness of MDHE and its generalization ability. As shown in Table III, we use some common evaluation metrics to evaluate MDHE's detection performance numerically, such as accuracy (ACC), precision (P), recall (R), F-measure (F).

TABLE III
PERFORMANCE EVALUATION METRICS

Term	Abbr	Definition
True Positive	TP	#malware samples are correctly classified as malwares
True Negative	TN	#benign samples are correctly classified as benign samples
False Negative	FN	#malware samples are misclassified as benign samples
False Positive	FP	#benign samples are misclassified as malwares
True Positive Rate	TPR	$TP/(TP+FN)$
False Positive Rate	FPR	$FP/(FP+TN)$
Accuracy	ACC	$(TP+TN)/(TP+TN+FN+FP)$
Precision	P	$TP/(TP+FP)$
Recall	R	$TP/(TP+FN)$
F-score	F	$2RP/(R+P)$
ROC curve	AUC	Area under ROC curve

TABLE IV
DETECTION PERFORMANCE OF THE MDHE VERSUS BASELINE MODELS FOR MALWARE DETECTION ON FOUR BENCHMARK DETASETS

Experimental dataset	CNN	DNN	GRU	LSTM	CapsNet
Drebin	95.42	97.48	96.36	97.10	97.66
MalDroid	95.18	97.18	97.12	96.64	97.26
AndroZoo	89.28	92.56	91.61	89.25	93.80
VirusShare	95.06	95.39	95.52	95.52	96.05

TABLE V
MDHE VERSUS OTHER PROPOSED SYSTEMS FOR MALWARE DETECTION

Experimental dataset	Permission [11]	API [28]	MDHE
Drebin	94.22	94.84	97.66
MalDroid	93.75	92.38	97.26
AndroZoo	83.86	88.62	93.80
VirusShare	95.20	93.94	96.05

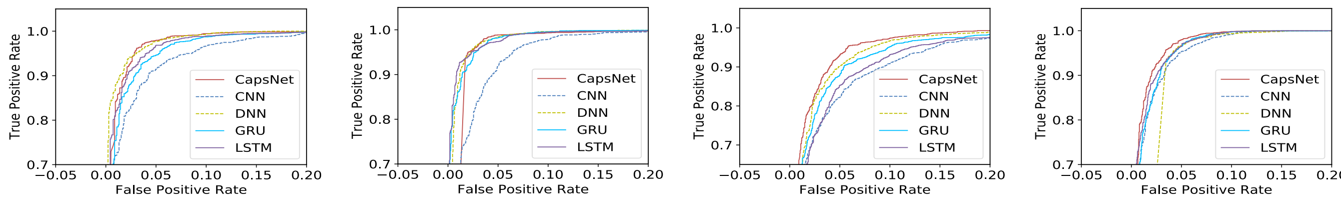
B. Malware Detection With Different Classifiers

In this part, we use five neural networks as the baseline models, which include Capsule Network (CapsNet) [29], Deep Neural Network (DNN) [30], Convolutional Neural Network (CNN) [31], Gated Recurrent Unit (GRU) [32] and Long Short Term Memory (LSTM) [33]. Moreover, Receiver Operating Characteristic (ROC) curve is used to quantify the detection performance of all baseline models on four benchmark datasets. Fig. 6 and Table IV summarize the comparison results. We can see that CapsNet achieves the best performance. Compared with the baseline model CNN, CapsNet abandons the pooling layer, thus requiring less training data to achieve the expected effect. Also, CapsNet relies on a dynamic routing mechanism to extract detailed features, in which the nonlinear compression function is used to compress the short vector almost to 0, while the length of the long vector is scaled up to approximately 1. As a result, the model can better detect subtle differences in data.

C. Method Comparison With Other Methods

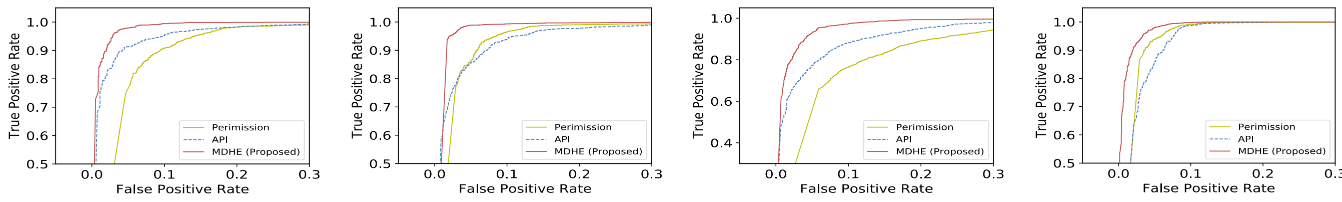
To verify the effectiveness of MDHE, we compared the detection performance of MDHE with the two state-of-the-art approaches, i.e., [11] and [28].

Authorized licensed use limited to: University of Science & Technology of China. Downloaded on January 27,2024 at 12:38:38 UTC from IEEE Xplore. Restrictions apply.



(a) Drebin dataset (b) MalDroid dataset (c) AndroZoo dataset (d) VirusShare dataset

Fig. 6. ROC curve of the MDHE versus baselines for malware detection on four benchmark datasets.

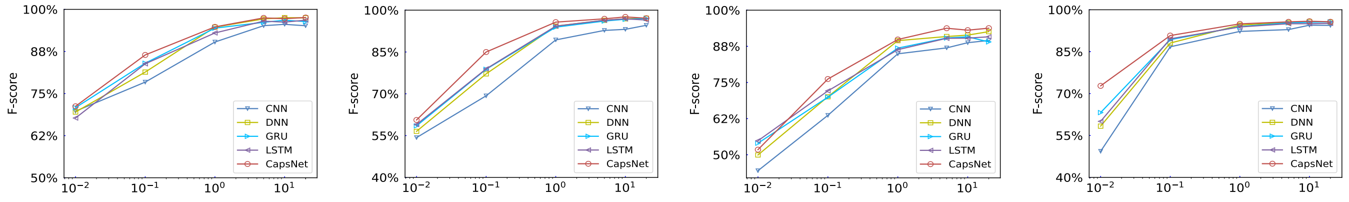


(a) Drebin dataset (b) MalDroid dataset (c) AndroZoo dataset (d) VirusShare dataset

Fig. 7. Experimental results of different feature comparisons.

TABLE VI
THE TRADE-OFF BETWEEN DATA UTILITY AND PRIVACY

	Drebin Dataset			MalDroid Dataset			AndroZoo Dataset			VirusShare Dataset		
MDHE	P	R	F-score	P	R	F-score	P	R	F-score	P	R	F-score
$\epsilon = 0.1$	83.63	89.49	86.46	82.90	87.18	84.99	72.50	80.28	76.19	87.91	92.38	90.09
$\epsilon = 5$	97.74	97.19	97.46	95.64	98.19	96.90	93.53	93.99	93.76	94.25	97.13	95.67
$\epsilon = 20$	98.29	96.87	97.57	96.80	97.44	97.12	93.28	94.27	93.78	94.70	96.69	95.69
noDP	97.98	97.35	97.66	97.62	96.89	97.26	93.00	94.60	93.80	95.46	96.65	96.05



(a) Drebin dataset (b) MalDroid dataset (c) AndroZoo dataset (d) VirusShare dataset

Fig. 8. Experimental results of different model comparisons.

- Perimission [11]: Arora et al. constructed a permission-based malware detection, and used machine learning models to identify malware.
- API [28]: Zou et al. proposed an API-based malware detection method, which uses a static analysis mechanism to extracted API features to describe the threat behavior of Android malware.

Fig. 7 and Table V show results comparing the detection accuracy of MDHE with that of two state-of-the-art methods [11] and [28]. As we expected, the results in Fig. 7 and Table V demonstrate that our MDHE consistently outperforms the two methods [11] and [28]. Typically, sensitive API calls are controlled by permissions, which can be used to access sensitive information (such as user's location or other personal information) and perform some sensitive tasks (such as changing WiFi status or sending messages). More specifically, malware tends to use some specific sensitive API calls, and thus malicious activities can be reflected by specific combinations of permissions and API calls. The *AUC* of the proposed MDHE achieves 0.99, which is better than the two methods [11] and [28]. This is because Arora et al. [11] only utilizes the permission features. Similarly, Zou et al. [28] only uses API features. As a result, our approach combines permissions and API features to help us analyze malware more effectively. This explains why our method achieves the best detection performance.

D. Evaluation on Privacy Budget ϵ

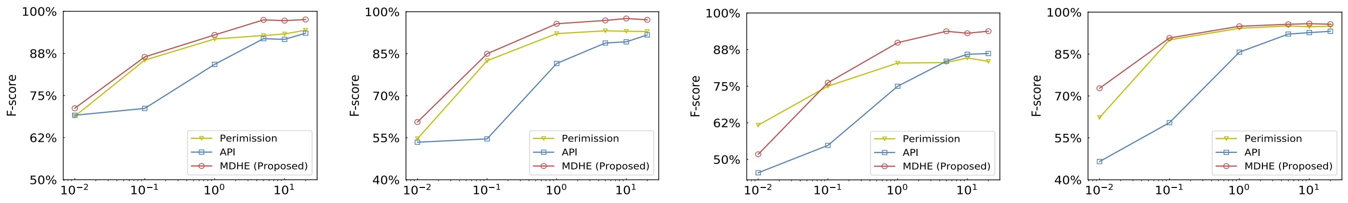
In this section, a series of experiments were conducted to validate the effectiveness of the proposed privacy attention mechanism in MHDE. Table VI presents the impact of different privacy budgets ϵ on detection performance of MHDE. When the value of ϵ is small, MHDE exhibits poor performance, indicating that excessive noise has been added into the input features, leading to a deterioration in model performance. As ϵ increases, the detection performance improves, demonstrating the effectiveness of our method in preserving privacy and utility.

To evaluate the effectiveness of DP, we conducted experiments on five classifiers, *i.e.*, CapsNet [29], CNN [31],

Authorized licensed use limited to: University of Science & Technology of China. Downloaded on January 27,2024 at 12:38:38 UTC from IEEE Xplore. Restrictions apply.

TABLE VII
MDHE Versus Baseline Models for Malware Detection

Models	Drebin Dataset				MalDroid Dataset				AndroZoo Dataset				VirusShare Dataset			
	ϵ=0.1	ϵ=5	ϵ=20	noDP	ϵ=0.1	ϵ=5	ϵ=20	noDP	ϵ=0.1	ϵ=5	ϵ=20	noDP	ϵ=0.1	ϵ=5	ϵ=20	noDP
CNN	78.40	95.15	95.13	95.42	69.28	92.74	94.57	95.18	63.63	86.96	89.52	89.28	86.74	92.90	94.40	95.06
DNN	81.37	97.16	97.49	97.48	77.15	96.36	97.11	97.18	70.05	90.89	92.58	92.56	88.08	95.42	95.57	95.39
GRU	84.01	96.15	96.36	96.36	78.80	96.09	97.01	97.12	69.92	90.38	89.03	91.61	89.29	95.34	95.66	95.52
LSTM	83.83	96.49	96.78	97.10	78.94	96.39	96.46	96.64	72.08	90.28	90.80	89.25	89.60	95.00	95.11	95.52
CapsNet	86.46	97.46	97.57	97.66	84.99	96.90	97.12	97.26	76.19	93.76	93.78	93.80	90.09	95.67	95.69	96.05

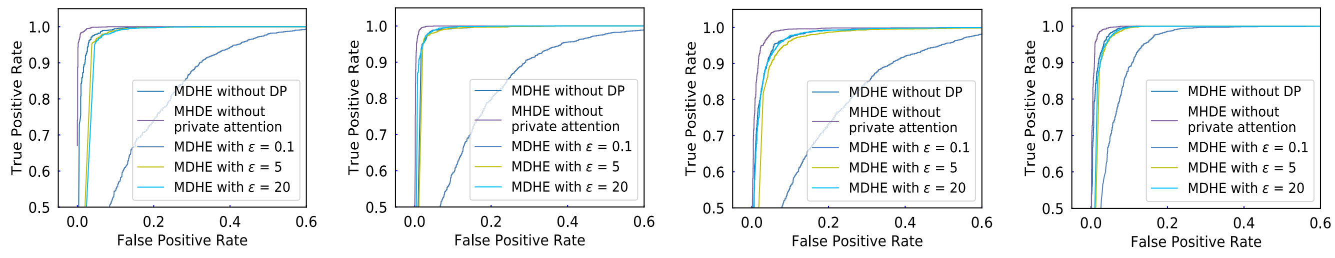


(a) Drebin dataset (b) MalDroid dataset (c) AndroZoo dataset (d) VirusShare dataset

Fig. 9. Experimental results of different feature comparisons.

TABLE VIII
Detection Performance Comparison With Other Proposed Systems

Methods	Drebin Dataset				MalDroid Dataset				AndroZoo Dataset				VirusShare Dataset			
	ϵ=0.1	ϵ=5	ϵ=20	noDP	ϵ=0.1	ϵ=5	ϵ=20	noDP	ϵ=0.1	ϵ=5	ϵ=20	noDP	ϵ=0.1	ϵ=5	ϵ=20	noDP
[11]	85.51	92.82	94.40	93.58	84.99	93.19	92.93	93.71	75.01	83.06	83.48	83.41	90.77	95.07	94.95	94.72
[28]	71.17	91.88	93.50	94.91	54.59	88.84	91.71	92.58	54.80	83.52	86.16	88.17	60.44	92.15	93.15	93.47
MDHE	86.46	97.46	97.57	97.66	84.99	96.90	97.12	97.26	76.19	93.76	93.78	93.80	90.09	95.67	95.69	96.05



(a) Drebin dataset (b) MalDroid dataset (c) AndroZoo dataset (d) VirusShare dataset

Fig. 10. Ablation studies of MDHE on the four benchmark datasets.

DNN [30], GRU [32], LSTM [33]. Fig. 8 and Table VII present the accuracy results of each model across varying privacy budgets ϵ. Notably, with an increase in ϵ, the performance of all baseline models saturates and exhibits notable improvement on all benchmark datasets. It is evident that various models exhibit distinct behaviors concerning their tolerance towards the amount of noise added to the input data.

Additionally, we assess the utility and privacy implications of the proposed privacy attention for different baseline methods, [11] and [28]. As shown in Fig. 9 and Table VIII, when ϵ increases (corresponding to less noise), the enhanced data utility enables more accurate learning for the two baseline methods. However, their performance is still not comparable to our MHDE. This observation emphasizes the importance of selecting an appropriate privacy budget to achieve a trade-off between data utility and privacy.

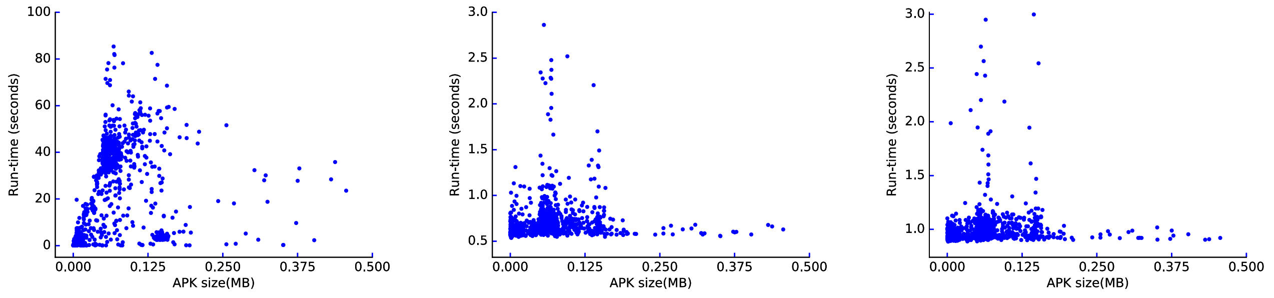
E. Effectiveness of Private Attention Module

In this section, different components should have different contributions to the detection performance. As shown in Fig. 10 and Table IX, we designed a series of baseline models to demonstrate the effectiveness of the private attention module used in the MDHE model, where the differential privacy (DP) approach is used to prevent inference attacks on users' data, and thus better protecting the privacy of users. We show the performance of MDHE under different privacy budgets ϵ. It can be seen that a smaller privacy budget (e.g. $\epsilon = 0.1$) results in lower detection accuracy because too much noise is introduced into the input data. In addition, MDHE can achieve better performance on the four benchmark datasets without using the DP mechanism (i.e., MDHE without DP).

TABLE IX
Effect of Different Components

Methods	MDHE			MDHE	MDHE without
	ϵ=0.1	ϵ=5	ϵ=20	without DP	private attention
Drebin	86.46	97.46	97.57	97.66	98.89
MalDroid	84.99	96.90	97.12	97.66	98.75
AndroZoo	76.19	93.76	93.78	93.80	95.74
VirusShare	90.09	95.67	95.69	96.05	97.22

Authorized licensed use limited to: University of Science & Technology of China. Downloaded on January 27,2024 at 12:38:38 UTC from IEEE Xplore. Restrictions apply.



(a) De-compilation and MSG construction (b) Private attention (c) Classification

Fig. 11. The runtime for processing one sample.

This shows that the introduction of DP reduces the final detection accuracy of the learning model, which is consistent with our expectations.

As mentioned earlier, MDHE decouples a large model into two parts. Although the proposed separation model effectively protects user privacy, it inevitably leads to model performance degradation. Therefore, we also evaluated MDHE performance without separation mode (i.e. MDHE without private attention). The model achieves optimal performance on four benchmark datasets. This verifies that model separation actually degrades detection performance. Moreover, we believe that a small performance penalty is acceptable because our model protects user privacy.

F. Processing Time Evaluation

In this part, we conduct experiments to evaluate the average run time of our method for detecting arbitrary malware. As previously noted, our method is mainly divided into three phases: 1)

1) **De-compilation and MSG construction.** The Dalvik code is decompiled from the APK file of a given application. Then, we construct the FCG to generate MSG using graph analysis.
2) **Private attention.** We use a DP-based attention mechanism to give higher weights to important features and then injected Laplace noise into the features.
3) **Classification.** We load the trained model on the device and classify each sample.

Fig. 11 displays the run time of each phase, with the X-axis representing the APK size for a given sample and the Y-axis representing the corresponding run-time. The construction of the MSG is affected by the APK size, as evident from Fig. 11 (a). This is because a larger APK file contains more sensitive APIs. It is evident that processing an MSG with thousands of nodes will take more time. In most cases, our method takes less than 100 seconds to process a single sample. Fig. 11 (b) and (c) demonstrate that private attention module and model classification are almost unaffected by the APK size.

Additionally, we measured the training and testing times of MDHE, and the results are presented in Table X. In our experiments, we found that the MDHE model achieves optimal performance after approximately 20 iterations. This implies that after a finite number of iterations, the model has successfully captured the underlying malicious patterns and features. As a result, training the MDHE takes less than 30 minutes. Although there is a slight increase in training time compared to CNN and DNN, the outstanding detection accuracy of MDHE more than compensates for its computational overhead, as evidenced by its consistently superior detection accuracy when compared to other baseline methods (see Section V-B for details). In addition to the commendable training efficiency, the time spent on model inference/testing is almost negligible. For example, the average inference time lies at just 13 ms for processing a single sample. This rapid inference speed is crucial in real-time malware detection as it enables quick response to effectively mitigate potential malware threats. Based on the above facts, we can confidently claim that our MDHE demonstrates exceptional performance and efficiency in malware detection.

TABLE X
EXECUTION TIME COMPARISON

Model	Training time		Inference / Testing time		
	#Train samples	Train time (per epoch)	#Testing samples	#Testing time	Avg. time (per sample)
DNN	20,000	31 s	5000	1.08 s	2 ms
CNN	20,000	31 s	5000	1.03 s	2 ms
GRU	20,000	84 s	5000	7.58 s	15 ms
LSTM	20,000	114 s	5000	9.30 s	18 ms
MDHE	20,000	78 s	5000	6.51 s	13 ms

VI. ENGINEERING APPLICATIONS

We apply our proposed MDHE to a city edge computing scenario with many edge servers and user devices (such as smart cars, smart phones and intelligent hospitals, etc.). As shown in Fig. 12, each edge server connects to a set of local user devices, is responsible for these user devices, typically covering an area, and performs timely data analysis. Before deploying the malware detection system, edge server screened a set of trusted training devices through TE mechanism, and trained a well-performing malware detection model. Moreover, the edge server deploys the feature generation module and private attention module on the users' end devices. When a user want to install a new app on the end device, the APK files of the app is first processed locally, then feature extraction and Laplacian noise injection are carried out. After that, the generated intermediate features are passed to the edge server

Authorized licensed use limited to: University of Science & Technology of China. Downloaded on January 27,2024 at 12:38:38 UTC from IEEE Xplore. Restrictions apply.

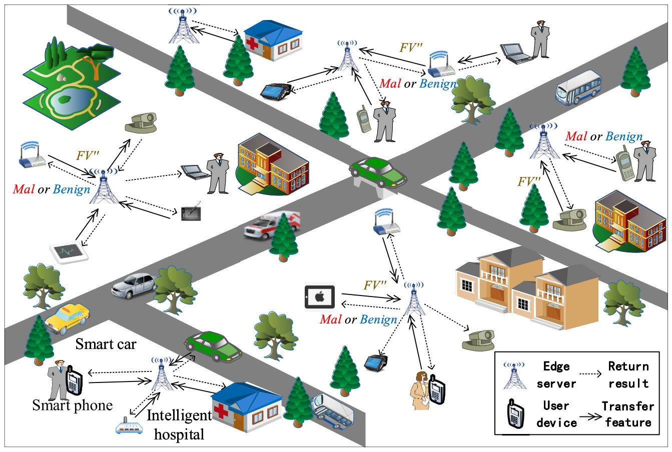


Fig. 12. Engineering applications of smart city.

covering the corresponding area. Finally, the edge server returns the result of model inference to the user, which indicates whether or not the app has malicious intent. If the app is malware, the user refuses to install it on the terminal device. In fact, the process is very fast. In our case, edge servers can protect user devices from malware attacks, while protecting users' privacy.

VII. Conclusion

In this paper, we propose a hybrid user-edge architecture (MDHE) for malware detection, which decomposes a large complex deep neural network into two parts and deploys them to the user device and the edge server respectively. The proposed MDHE can effectively detect malware while protecting user's privacy. Then, we deploy a feature generation module and a private attention module on the user device, where the feature generation module is used to generate features, and the attention mechanism is used to extract key features. Finally, the malware classifier is deployed on edge server to reconstruct the disturbed features and detect malware. The results show that our method has strong generalization ability, and is better than other existing methods. Finally, our method can protect the privacy of users' sensitive data. In the future, we can design a end-edge-cloud architecture to improve the accuracy of malware detection. Moreover, we consider combining dynamic information (such as network flow and data flow) to detect malware.

References

[1] 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.

[2] G. Zhang et al., "TSDroid: A novel Android malware detection framework based on temporal & spatial metrics in IoMT," *ACM Trans. Sensor Netw.*, vol. 19, no. 3, pp. 1–23, Aug. 2023.

[3] X. Deng, J. Zhu, X. Pei, L. Zhang, Z. Ling, and K. Xue, "Flow topology-based graph convolutional network for intrusion detection in label-limited IoT networks," *IEEE Trans. Netw. Service Manag.*, vol. 20, no. 1, pp. 684–696, Mar. 2023.

[4] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *Proc. ICDCS*, 2017, pp. 328–339.

[5] Y. Kang et al., "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGARCH Comput. Archit. News*, vol. 45, no. 1, pp. 615–629, 2017.

[6] 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.

[7] X. Pei, X. Deng, S. Tian, L. Zhang, and K. Xue, "A knowledge transfer-based semi-supervised federated learning for IoT malware detection," *IEEE Trans. Dependable Secure Comput.*, vol. 20, no. 3, pp. 2127–2143, May/Jun. 2023.

[8] P. Jiang, X. Deng, L. Wang, Z. Chen, and S. Zhang, "Hypergraph representation for detecting 3D objects from noisy point clouds," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 7, pp. 7016–7029, Jul. 2023.

[9] X. Deng, X. Pei, S. Tian, and L. Zhang, "Edge-based IIoT malware detection for mobile devices with offloading," *IEEE Trans. Ind. Informat.*, vol. 19, no. 7, pp. 8093–8103, Jul. 2023.

[10] X. Chen et al., "CruParamer: Learning on parameter-augmented API sequences for malware detection," *IEEE Trans. Inf. Forensics Security*, vol. 17, pp. 788–803, 2022.

[11] A. Arora, S. K. Peddoju, and M. Conti, "PermPair: Android malware detection using permission pairs," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 1968–1982, 2020.

[12] R. Sun et al., "Learning fast and slow: Propedeutica for real-time malware detection," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 6, pp. 2518–2529, Jun. 2022.

[13] Y. Chai, L. Du, J. Qiu, L. Yin, and Z. Tian, "Dynamic prototype network based on sample adaptation for few-shot malware detection," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 5, pp. 4754–4766, May 2023.

[14] J. Xu, Y. Li, R. H. Deng, and K. Xu, "SDAC: A slow-aging solution for Android malware detection using semantic distance based API clustering," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 2, pp. 1149–1163, Mar. 2022.

[15] H.-J. Zhu, L.-M. Wang, S. Zhong, Y. Li, and V. S. Sheng, "A hybrid deep network framework for Android malware detection," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 12, pp. 5558–5570, Dec. 2022.

[16] J. Jeon, B. Jeong, S. Baek, and Y.-S. Jeong, "Hybrid malware detection based on Bi-LSTM and SPP-net for smart IoT," *IEEE Trans. Ind. Informat.*, vol. 18, no. 7, pp. 4830–4837, Jul. 2022.

[17] D. Tian et al., "MDCD: A malware detection approach in cloud using deep learning," *Trans. Emerg. Telecommun. Technol.*, vol. 33, no. 11, p. e4584, Nov. 2022.

[18] P. Mishra et al., "VMShield: Memory introspection-based malware detection to secure cloud-based services against stealthy attacks," *IEEE Trans. Ind. Informat.*, vol. 17, no. 10, pp. 6754–6764, Oct. 2021.

[19] K. Vahedi and K. Afhamisisi, "Cloud based malware detection through behavioral entropy," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2021, pp. 6046–6048.

[20] W. Tang, J. Ren, and Y. Zhang, "Enabling trusted and privacy-preserving healthcare services in social media health networks," *IEEE Trans. Multimedia*, vol. 21, no. 3, pp. 579–590, Mar. 2019.

[21] W. Tang, J. Ren, K. Zhang, D. Zhang, Y. Zhang, and X. Shen, "Efficient and privacy-preserving fog-assisted health data sharing scheme," *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 6, pp. 1–23, Nov. 2019.

[22] M. Blaze, J. Feigenbaum, and J. Lacy, "Decentralized trust management," in *Proc. IEEE Symp. Secur. Privacy*, May 1996, pp. 164–173.

[23] V. Toubiana, A. Narayanan, D. Boneh, H. Nissenbaum, and S. Barocas, "Adnostic: Privacy preserving targeted advertising," in *Proc. NDSS*, 2010, pp. 1–23.

[24] M. Shahpasand, L. Hamey, D. Vatsalan, and M. Xue, "Adversarial attacks on mobile malware detection," in *Proc. IEEE 1st Int. Workshop Artif. Intell. Mobile*, Feb. 2019, pp. 17–20.

[25] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "AndroZoo: Collecting millions of Android apps for the research community," in *Proc. 13th Int. Conf. Mining Softw. Repositories*, May 2016, pp. 468–471.

[26] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "Drebin: Effective and explainable detection of Android malware in your pocket," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2014, pp. 23–26.

[27] 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., Int. Conf Pervasive Intell. Comput., Int. Conf Cloud Big Data Comput., Int. Conf Cyber Sci. Technol. Congr. (DASC/PiCom/CBDCom/CyberSciTech)*, Aug. 2020, pp. 515–522.

[28] D. Zou et al., "IntDroid: Android malware detection based on API intimacy analysis," *ACM Trans. Softw. Eng. Methodol.*, vol. 30, no. 3, pp. 1–32, 2021.

Authorized licensed use limited to: University of Science & Technology of China. Downloaded on January 27,2024 at 12:38:38 UTC from IEEE Xplore. Restrictions apply.

[29] S. Wang, G. Zhou, J. Lu, and F. Zhang, "A novel malware detection and classification method based on capsule network," in *Proc. ICAIS*, vol. 11632, 2019, pp. 573–584.

[30] 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 Security*, vol. 14, no. 3, pp. 773–788, Mar. 2019.

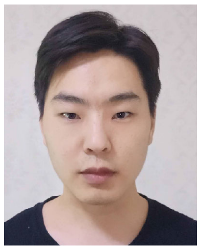
[31] N. Lachtar, D. Ibdah, and A. Bacha, "Toward mobile malware detection through convolutional neural networks," *IEEE Embedded Syst. Lett.*, vol. 13, no. 3, pp. 134–137, Sep. 2021.

[32] 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. (ICASSP)*, Mar. 2017, pp. 2482–2486.

[33] M. Sewak, S. K. Sahay, and H. Rathore, "LSTM hyper-parameter selection for malware detection: Interaction effects and hierarchical selection approach," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2021, pp. 1–9.

Xiaoheng Deng (Senior Member, IEEE) received the Ph.D. degree in computer science from Central South University, Changsha, Hunan, China, in 2005. Since 2006, he has been an Associate Professor and then a Full Professor with the Department of Communication Engineering, Central South University, where he is currently a Joint Researcher with the Shenzhen Research Institute. He is a Senior Member of CCF, a member of CCF Pervasive Computing Council, and a member of ACM. He has been the Chair of CCF YOCSEF CHANGSHA from 2009 to 2010. His research interests include network security, edge computing, the Internet of Things, online social network analysis, data mining, and pattern recognization.

Haowen Tang was born in Changsha, Hunan, China, in 1998. He received the B.Sc. degree in information security from Central South University, Changsha, in 2019, where he is currently pursuing the M.Sc. degree with the School of Computer Science and Engineering. His major research interests are the IoT security and edge computing.



Xinjun Pei (Student Member, IEEE) is currently pursuing the Ph.D. degree with 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 the IoT security.

Deng Li (Member, IEEE) received the B.S. degree in computer science from Hunan University, Changsha, China, in 1999, and the M.S. and Ph.D. degrees in computer science from Central South University, Changsha, in 2002 and 2008, respectively. He is currently an Associate Professor with the School of Computer Science and Engineering, Central South University. His current research interests include autonomic communications, mobile computing, and the Internet of Things.

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 Ph.D. degree from the Department of Electronic Engineering and Information Science (EEIS), USTC, in 2007. From May 2012 to May 2013, he was a Post-Doctoral Researcher with the Department of Electrical and Computer Engineering, University of Florida. Currently, he is a Professor with the School of Cyber Science and Technology, USTC. His research interests include next-generation internet architecture design, transmission optimization, and network security. He serves on the Editorial Board Member for several journals, including IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS, and IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT. He has also served as a (Lead) Guest Editor for many reputed journals/magazines, including IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, *IEEE Communications Magazine*, and *IEEE Network*.

Authorized licensed use limited to: University of Science & Technology of China. Downloaded on January 27,2024 at 12:38:38 UTC from IEEE Xplore. Restrictions apply.