TransMalDE: An Effective Transformer Based Hierarchical Framework for IoT Malware Detection

Xiaoheng Deng, *Senior Member, IEEE*, Zhe Wang, Xinjun Pei, *Student Member, IEEE*, and Kaiping Xue, *Senior Member, IEEE*

***Abstract*—With the rapid development of the Internet of Things (IoT) and cloud applications, cloud service providers have rented out access to servers to IoT devices for computing and storage purposes, providing users with a variety of services and functionality. The prevalence of malware attacks against IoT devices has led to serious and critical concerns with respect to cyber security. In response to this growing threat, many IoT security providers are adopting cloud-based, centralized malware detection systems. However, this may cause back-and-forth communication, which violates the real-time requirement of malware detection. The ever-growing edge computing has resulted in the development of new and more efficient data processing. By exploiting the proximity benefits and the computation capacity of edge computing, we propose a hierarchical IoT malware detection framework (namely TransMalDE) to migrate user computation-intensive malware detection tasks to neighboring edge computing nodes, which improves the efficiency of malware detection. Moreover, considering the rigidity of the current network infrastructure and the complexity of AI-enabled malware detection tasks, we construct a Transformer-based detection model to capture the latent behavioral patterns of evolving malware attacks. Experimental results show that our TransMalDE consistently outperforms the existing state-of-the-art systems in malware detection on four benchmark datasets.**

***Index Terms*—Internet of things, malware detection, edge computing, security systems.**

Manuscript received 28 October 2022; revised 14 April 2023; accepted 2 July 2023. Date of publication 6 July 2023; date of current version 8 January 2024. This work was supported in part by the National Natural Science Foundation of China Project under Grants 62172441 and 61772553, in part by the National Natural Science Foundation of Hunan Province under Grant 2023JJ30696, in part by the Local Science and Technology Developing Fundation guided by Central Goverment (Free exploration project 2021Szvup166), in part by the Key Project of Shenzhen City Special Fund for Fundamental Research under Grant 202208183000751, and in part by the Opening Project of State Key Laboratory of Nickel and Cobalt Resources Comprehensive Utilization under Grants GZSYS-KY-2022-018 and GZSYS-KY-2022-024. Recommended for acceptance by Dr. Xuelong Li. *(Corresponding authors: Xiaoheng Deng; Xinjun Pei.)*

Xiaoheng Deng is 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).

Zhe Wang and Xinjun Pei are with the School of Computer Science and Engineering, Central South University, Changsha 410083, China (e-mail: 204711076@csu.edu.cn; pei_xinjun@163.com).

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/TNSE.2023.3292855

I. Introduction

REMARKABLY, using smartphone to access the Internet has become the norm in people's daily lives, which can be seen almost everywhere. The popularity of IoT [1] attracted widespread attention from malware developers. The past several years have witnessed a burgeoning software development business in online app marketplaces, such as the Google Play Store. Under the circumstances, Android malware detection systems have gained a significant role in IoT network security. Anti-malware providers strongly prefer scalable and lightweight methods to cope with the numbers of apps [2].

An increasing number of IoT services can be remotely conducted through Android platforms. Meantime, untrustworthy third-party markets provide many smart IoT applications for some specific IoT scenarios, such as various smart home services to control lightings and thermostats. IoT applications are growing phenomenally by powering a vast spectrum of the Android-based IoT devices. In IoT network, an operating system of a capable hardware device manages and controls massive number of sensors. Applications deployed on user devices send data and instructions to other devices or edges, providing convenient personalized services to users [3]. For example, the Android-based IoT devices gather information from all the connected remote smart devices and sensors, and upload data to the edge data center.

Another topic of discussion is that existing detection systems are not proving satisfactory to combat the sophisticated and well-designed IoT malwares [4]. Although traditional analytics methods like cloud-based centralized analytics [5], [6], [7], [8], [9] can discover threats by continuous monitoring of the system, in many cases, there is not enough time to prevent the occurrence of the threats on Android Things. In comparison, edge computing is closer to the terminals, which can provide lower latency. In this new scenario, applying deep learning to the edge intelligence for service deployment has become an ideal solution. It is clear that edge computing is able to analyze IoT data in near real-time. The edge nodes are most likely to perform reaction and predictive analytics, while the cloud farther from the end users performs model training on large-scaled dataset due to higher computing power. Thus, large-scale deep learning models are more suitable to be created and trained in the cloud to help the threat detection. Specifically, the detection

model will be retrained periodically to update its detection capabilities. However, uploading applications from unauthenticated unofficial markets or third-party resources directly to the cloud for security verification is also a time-consuming task due to the high network transmission overhead [10]. Therefore, it is reasonable to design a more effective malware security system in edge computing to improve the efficiency of malware detection.

Besides, modeling behavior information of malware is a fundamental topic in malware detection. Existing efforts on malware detection have focused on various static features, such as behavioral-based features [5] and API-based features [9], [11], [12]. Azmoodeh et al., [13] proposed an Internet of Battlefield Things (IoBT) malware detection system to extract and learn the OpCodes features, and used a deep learning model to classify malicious and benign applications. Sun et al., [14] proposed a behavioral-based malware framework that realizes a two-phase detection mechanism to identify malware. However, most of these methods cannot fully represent the latent behavioral information contained in the static features while deriving a new set of features is rather difficult. Although shallow machine learning models are widely used in malware detection, it is challenging to cope with the continuously evolving IoT malware without exploring the latent information. This motivates us to develop new detection model to capture the program semantic of malware. In this article, we obtain the behaviors information (i.e. API) of malware provided by static analysis methods, which had demonstrated good performance in the literature and are relatively easy to obtain.

In this article, we design a hierarchical IoT malware detection framework (TransMalDE) to ensure that the services provided to users are safe, reliable, and credible. This framework can keep the malware detection problem sufficiently tractable, where the edge computing nodes are used for performing the computation-intensive tasks initiated by user devices to improve the efficiency of malware detection. This design is sufficient to ensure the security of the IoT applications deployed on the users' devices. Specifically, in this framework, the cloud is designed to train a large-scale deep learning model due to its powerful computing power. Moreover, we use a lightweight feature extraction to extract sensitive subgraph features to reflect the latent behavior patterns of malware, while reducing the computing overhead of Mobile-IoT devices. Finally, we construct a Transformer-based detection model to extract the textual semantic patterns from malware. The fact that the superior performance of the Transformer-based IoT malware detection method indicates that the Transformer model is a worthwhile exploration.

Our major contributions are as follows:

- We propose a hierarchical IoT malware detection framework, which can provide a safe and trusted environment for the deployment of malware detection, while improving the efficiency of malware detection by migrating computation-intensive tasks to neighboring edge computing nodes.
- We propose a sensitive subgraph-based graph mining approach that can reflect the latent behavior patterns of malware. Moreover, we construct a Transformer-based detection model to extract the textual semantic patterns from malware for modeling higher-level semantic concepts and facts in malware.
- Experimental results on four benchmark dataset demonstrate that the proposed TransMalDE reduces the latency and achieves substantial improvement over various neural model baselines.

In the rest of the article, we overview the related work in Section II. And then we describe the security model and adversary model in Section III. Section IV gives the edge-based security system in detail. In Section V, we evaluate the proposed system and analyze the experimental results. Finally, Section VI concludes the article.

II. Related Work

The popularity of IoT networks attracted malicious adversaries to constantly generate and evolve new IoT malware, which can be hardly detected by most existing security products [13], incurring serious security concerns for users to deploy Mobile-IoT applications in Android-based IoT devices. For Android malware detection, a typical approach is the Opcode-based static analysis provided by anti-malware providers. Amin et al. extracted Opcode-based features from the Android Package Kit bytecode, which was then fed into deep neural networks for malware detection. Specifically, they used a variety of neural network models to detect malware, such as generative adversarial networks [15], bidirectional long short-term memory (BiLSTMs) [16], and fully connected networks (FCN) [17]. Moreover, Aidin et al. [18] proposed a novel watermarking algorithm for the dynamic authentication of IoT signals to detect cyber-attacks. Liu et al. [19] proposed a reflection backdoor (Refool) approach based on the mathematical modeling of physical reflection models, which implanted reflections as a backdoor into a victim model [19], [20]. Quiring et al. [21] proposed a new scheme to hide traces of operation and overlays of clean-label poisoning by combining poisoning and image-scaling attacks. Rasthofer et al. [22] proposed a FuzzDroid framework that combines static and dynamic analyses to automatically generate an execution environment. In this environment, the hidden malicious behaviors of malware are exposed. However, they may fail in complex situations. Malware developers often adopt various obfuscation techniques to evade detection by the analysis system. The obfuscation techniques require more modification of its bytecode, strings, or resource files compared to the past, making applications more difficult to be analyzed by manual or automated tools. In this article, we develop a lightweight feature extraction model and evaluate the effectiveness of our system against obfuscated malware on a wide variety of datasets.

Recently, various graph analysis techniques have been proposed for malware detection, such as Opcode graph, frequent sub-graph, control flow graph (CFG), and function call graph (FCG). Moreover, Fairbanks et al. [23] used a graph machine learning approach to process the CFG. Li et al. [11] proposed a hybrid feature encoder to capture semantic information from different types of API arguments. In [24], Wu et al. reinterpreted opcode sequences to unify user-defined function calls, and then used the Graph2vec method to capture the semantics

TABLE I
COMPARISON BETWEEN MALWARE DETECTION METHODS

Items	Years	Feature	Detection	Model	Category	Performance
[11]	2022	API call graph	-	DMalNet	Static analysis	Low
[23]	2021	CFG	-	Graph-based Model	Static analysis	Low
[24]	2023	FCG	-	Graph2vec	Static analysis	High
[25]	2022	FCG	-	Machine learning	Static analysis	Low
[12]	2021	API graph	-	SDGNet	Static analysis	High
[8]	2020	Signatures	In-cloud	TrustSign	Dynamic analysis	High
[6]	2021	Performance metrics	In-cloud	CNN	Dynamic analysis	Low
[5]	2021	Behavior features	In-cloud	Behavioral similarity	Dynamic analysis	Low
[7]	2022	Dynamic features	In-cloud	MDCD	Dynamic analysis	High
[9]	2021	API call graph	In-cloud	DMalNet	Dynamic analysis	High
[26]	2021	Permission and API calls	On-device	Broad learning	Static analysis	Low
[27]	2021	Binary features	On-device	MobiTive	Static analysis	Low
[28]	2019	Funsion features	On-device	XGBoost	Static analysis	Low
Our work	-	Sensitive subgraph	Cloud-edge-end	Transformer	Static analysis	High

of the function-call relationships. By using a weighted graph matrix normalization method, Zhang et al. [12] converted the graph adjacency matrix into three symmetric graph matrices describing different aspects of node information interaction. Then, they proposed a behavior feature-based malware detection method that learns node information interactions from the three symmetric graph matrices. Jia et al. [25] proposed a two-stage learning approach to solve Open Set Recognition (OSR) problems. They first extracted function call graphs (FCGs) and then used a detransformation autoencoder (DTAE) to detect the malware. However, the above solution requires analyzing the entire CFG/FCG with thousands of nodes, which is clearly computationally more expensive and complex. To cope with this challenge, this artilce proposes a sensitive subgraph-based graph mining approach, which can reduce the graph's size while learning the latent behavior patterns of malware. Table I lists some representative malware detection methods.

Most existing studies on malware detection have discussed the cloud-based malware detection systems, which move heavy computing tasks to the cloud. Vahedi et al. [5] first extracted behavioral features from malware files, and then sent them to a cloud-based malware detection platform for the behavioral similarity test against known malware families. To defend against malware attacks in the cloud, Kimmell et al. [6] proposed cloud-based online malware detection, which runs different malware families in the cloud to collect process-level features. In [7], the authors implement a dynamic malware detection solution for cloud environments that uses virtualization-based techniques to collect runtime utilization and memory object information from the target virtual machine (VM). Nahmias et al. [8] used the cloud's virtualization technology to analyze malicious processes. During this process, the malware is unaware. Similarly, Mishra et al. [9] collected the runtime behavior of malware by performing virtual memory introspection from the hypervisor. However, these cloud-based malware detection methods may not have enough time to prevent the occurrence of malware attacks on IoT devices, and cannot meet the real-time requirements of IoT users for malware detection.

Moreover, there are some on-device malware detection studies. For on-device malware detection, the deep learning-based malware detectors are trained on servers and then transplanted to mobile devices. Yuan et al. [26] proposed a broad learning-based lightweight on-device malware detection method, which uses one-shot computation for model training. Feng et al. [27] proposed an effective malware detection system, which is deployed on mobile devices to provide real-time and responsive detection. Niu et al. [28] proposed a fusion feature-based malware detection model to detect malware attacks on IoT devices in autonomous driving. However, on-device malware detection is hard to implement due to the limited resources of mobile devices. Inspired by edge computing, this article considers moving heavy computing tasks to edge to deal with security issues. Edge computing can perform reactive and predictive analytics faster than the cloud, while also alleviating the lack of computing resources for IoT devices.

III. SYSTEM ARCHITECTURE AND THREAT MODEL

A. System Architecture

Edge security is an important guarantee for edge computing, which can improve the ability of edge to resist various security threats. There are many researches on the security aspects of edge, but most of which lay more emphasis on the network and routing security with none specifically targeting malware in edge computing ecosystem. We provide a holistic view of a hierarchical security framework for defending against malware, which consists of IoT devices, edge computing nodes and the cloud. It provide a safe and trusted environment for the development of edge computing. Fig. 1 shows the primary security handling of each layer of this structure. The detailed classification description is explained in Section IV.

1) The First Layer - IoT Devices: The edge network contains numerous end devices that are often programmed to complete a specific simple task. Here there are a set of applications and services at a given smart device. Malicious applications can be widely run on various IoT devices distributed geometrically to achieve the malicious intents. Therefore, our goal is to collect the applications, and then upload them to the upper layer for processing.

2) The Second Layer - Edge Computing Nodes: This layer consists of edge computing nodes, each of which performs computing tasks. Obviously, these computing nodes have significantly more computing power than end devices. Additionally, in comparison to the cloud, the analytics and detection system deployed on the edge server is able to take steps to prevent the occurrence of the issue in near real- time. These computing nodes

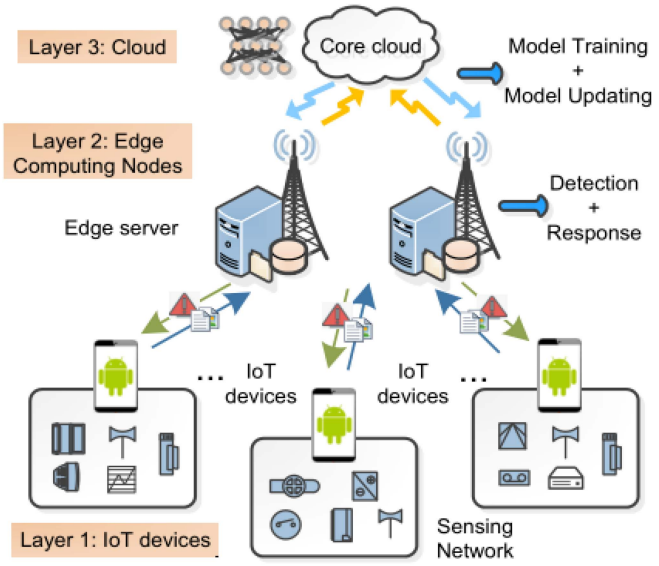


Fig. 1. The proposed hierarchical IoT malware detection framework.

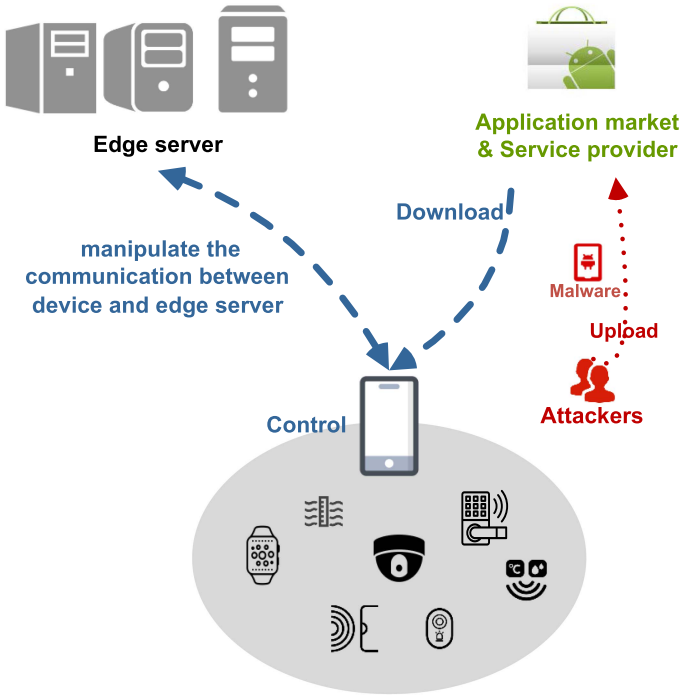


Fig. 2. An example of security analysis.

are used for performing the computing task requests initiated by edge devices. Furthermore, the detection model is deployed on this layer to help in execution. This means that edge servers, as the core of checking service of the received computing requests, either automatically or by user submission, from the end devices, should be paid more attention to have a greater impact. This layer is used to identify threat patterns on the data streams input from end devices by continuous system monitoring. It categorizes suspicious behavior and responds to deployment requests of applications. The goal is to return detection results to the end device for classification and alert. Furthermore, the quick feedback control mechanism will make a local reflex decisions to control the infrastructure, once suspicious applications and services are detected by our system.

3) The Third Layer - Cloud: Cloud computing can be used for large-scaled behavior analysis and threat detection. Complex deep learning models are more suitable for construction and training at this layer due to the powerful computing performance. Meanwhile, it is necessary to establish a virus database, which can help edge computing nodes to quickly query and determine the threat level. It is able to aggregate the statistics information from the edge of the network, train a large-scale deep learning model, and perform a more powerful analysis. In addition, in the case of large-scale service interruption of edge network (i.e. edge nodes cannot work under cyber-attacks and other network anomalies), the cloud is able to support response and resource management.

B. Threat Model

Therefore, mobile IoT applications play an important role in the IoT network, and their security needs to be guaranteed by service providers. Fig. 2 gives an example of security analysis. We list possible malware threats against the IoT network.

- *T1:* IoT applications can not only control and operate purchased IoT devices, but even provide them with Internet connectivity, such as cameras, smart door locks, smart watches and other smart devices. Adversaries may write malicious codes and infringe on the legitimate rights and interests of users.
- *T2:* Companion applications control and operate IoT devices through low-power wireless communication (eg, Bluetooth), and communicate with the cloud through Internet connections provided by smart mobile devices. Adversaries rely on the companion applications to control smart IoT devices. This may lead to the blindness of IoT devices to edge servers, when malware deliberately blocks certain messages.
- *T3:* The IoT devices have limited in team of computational power, memory space, etc. Attackers can plan an attack to any region in the edge network, and malware cannot be processed quickly by these limited devices.
- *T4:* Without explicitly prompting the user or obtaining permission, malware could be performed on IoT devices, stealing and analyzing users' personal information, secrets and privacy data.

As a result, a security protection system against malware needs to be established to protect the edge infrastructures, networks, applications, privacy data, and so on.

C. Security Analysis

IoT networks are now facing more targeted malware attacks. Because adversaries can leverage third-party IoT applications deployed on IoT devices to access critical external physical system controllers or create rules to control sensors to interact with servers. To prevent malware attackers from attacking IoT infrastructure, such as unauthorized access, Trojan attacks, device hijacking, etc., this article builds a hierarchical IoT malware detection framework to continuously monitor the IoT network

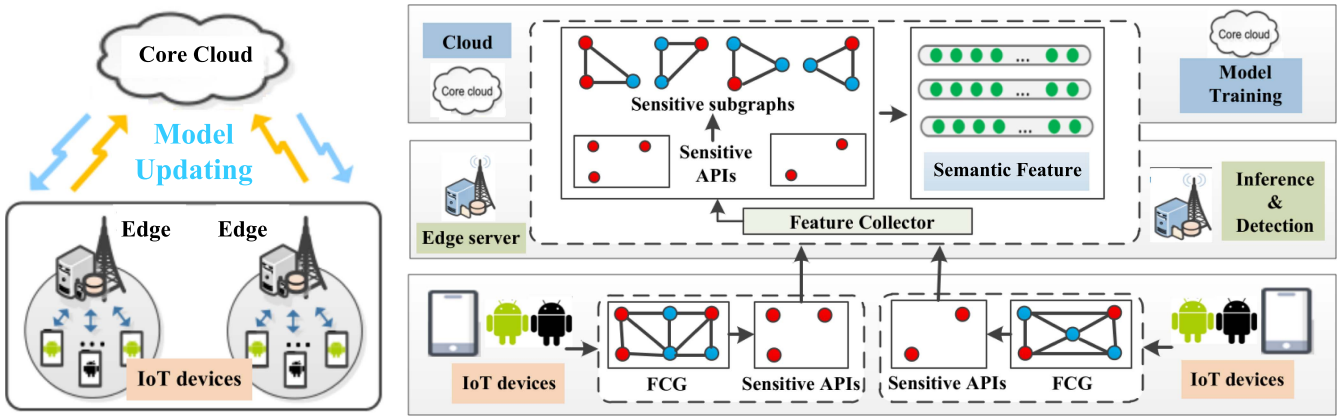


Fig. 3. An overview of our framework.

to detect malware threats. This framework extends the malware detection system to the cloud-edge-end architecture. In this framework, the edge computing platform provides computing services through an open API interface, which can be used for resource access authorization, data collection, and analysis. It can improve the efficiency of malware detection by migrating computation-intensive detection tasks to neighboring edge computing nodes. Specifically, we use the cloud computing platform to train a large-scale deep learning model, which is then deployed on the edge computing platform for quicker malware detection and turnaround.

IV. Our Proposed Detection Scheme

In this section, we propose a novel deep learning framework (TransMalDE), which extracts sensitive API call features. Then, we introduced word embedding (word2vec) technology to extract the potential semantic patterns from the set of sensitive API calls, which are then fed to the classification model for malware detection. The framework conducts three major processes; feature extraction, semantic vector generation and classification.

A. Static Code Analysis on IoT Device

There is a set of applications on a given smart device. The device is in charge of collecting and pre-processing the Android app files. The goal is to identify potential malicious behaviors and patterns from applications by continuously monitoring applications. In such a case, our system only collects the features specified by the feature extractor. Fig. 3 shows the system model. A topic worth discussing is how we extract malware domain knowledge from IoT applications. Due to the complexity of malware, it is imperative to develop a well-designed malware representation to obtain good malware detection performance. To achieve this, in our case, we build a function call graph (FCG) [29] by utilizing a disassembly tool (Androguard) and a graphical visualization software (Gephi), which extracts the callers and the callees from the Dalvik code of an given IoT application. Then, we give the definition of FCG as follows.

Definition 1. FCG: Let $G = (\mathcal{V}, \mathcal{E})$ be a FCG, where V represent the set of N nodes, and $\mathcal{E}$ represent the set of M edges. The node $v_i \in \mathcal{V}$ in the FCG corresponds to a function in a given IoT application. The edge $e_{i,j} \in \mathcal{E}$ represents the calling relation between the caller function v_i and the callee function v_j.

Specifically, FCG can reflects the program behaviors. However, it is clear that housands of nodes can be found in FCG, resulting in inefficient FCG analysis. In practice, we found that the function call graph (FCG) contained some sensitive API calls, which implies the potentially malicious behaviors of malware. This helps distinguish malware from benign apps. Moreover, Fan et al., [30] shows that different malware in the same family, even if their codes may be obfuscated, often invoked sensitive API calls via similar patterns. For example, multiple sensitive API calls are sequentially invoked in the different methods of different malware. Following this work [30], we designed a lightweight static code analysis mechanism to extract the sensitive API nodes (S_{API}) [31] and their corresponding neighbors from the FCG, which can reflect the specific behavior patterns of malware. Then, we define a set of sensitive API calls as follows.

$$API_{sen} = \{api_1, api_2, \ldots, api_n\}. \tag{1}$$

This strategy allows for obtaining good performances while minimizing the manual intervention and the computation overheads of battery-powered devices.

B. Detection System on Edge Computing Node

Our system is mainly used to check applications and issue the corresponding permissions or warnings to users. It is clear that edge computing is able to analyze IoT data in near real-time. The edge nodes are most likely to perform reaction and predictive analytics. Therefore, deep learning-based detection system is deployed on the edge computing node. We make the necessary modification for the centralized mechanism to fit it on the real applications. In our case, the edge server will check whether a given application has malicious intent. As shown in Fig. 3, our system primarily consists of data management module, learning and analysis module, and alarm module. The data management module first collects and stores the API features extracted by the IoT devices distributed geospatially, and then transforms them into semantic vectors. By doing this, the learning and analysis module fed the semantic vertors into the detection model to perform the malware detection, and give the inference results. After the detection process, the alarm module will transmit an

Algorithm 1: Semantic Vector Generation.

Input: IoT applications
Output: Semantic embedding vectors
Construct a FCG for a given app;
Create a sensitive API call sequence $API_{sen} = \{api_1, api_2, \ldots, api_n\}$ from FCG;
// Create a sensitive subgraph sequence $F_{SSubG} = \{f_{s_1}, f_{s_2}, \ldots, f_{s_n}\}$ using (2)–(5);
for each $api_i \in API_{sen}$ **do**
  $ssubg_i \leftarrow \emptyset$;
  for each $v_k \in V$ **do**
    if $\mathrm{dis}(api_i, v_k) \leq k$ **then**
      $ssubg_i \leftarrow \{api_i\}$;
    end if
  end for
  $SSubG \leftarrow ssubg_i$;
  $f_{s_i} = I(ssubg_i) \cdot m(ssubg_i)$;
  $F_{SSubG} \leftarrow f_{s_i}$;
end for
$F \leftarrow \emptyset$ // Create semantic embedding vectors.
for each $f_i \in F_{SSubG}$ **do**
  Obtain a semantic embedding vector $f_i \leftarrow \{Emb(f_{s_i})\}_{i=1}^{N}$;
  $F \leftarrow f_{s_i}$;
end for
return $F = \{f_1, f_2, \ldots, f_n\}$.

alarm message to the users if there is malware. Then, an IoT device is convinced that the designated application is safe based on the results.

1) Sensitive Subgraph Representation on Edge Computing Node: In this part, we introduce the TF-IDF technology to pay different attention to sensitive APIs and enhance feature representation. Intuitively, the sensitivity of a sensitive API api_i is positively related to the amount of malware that calls it. We define the maliciousness degree as follows.

$$M_{api} = \{m(api_1), m(api_n), \ldots, m(api_n)\}, \tag{2}$$

where $m(api_i)$ is the maliciousness degree of $i\text{-}th$ sensitive API api_i. Let $|Mal|_{total}$ and Num_b be the total number of malware and benign apps that have called the sensitive API node api_i, respectively. Let $|Mal|_{api_i}$ and $|Ben|_{api_i}$ be the number of malware and benign apps that have called the sensitive API node api_i, respectively. We have:

$$m\left(api_i\right) = P_m\left(api_i\right) * \log \frac{|Mal|_{total} + |Ben|_{total}}{|Mal|_{api_i} + |Ben|_{api_i}} \tag{3}$$

where $P_m(api_i)$ represents the percentage of malware that calls api_i. Then, we define a set of sensitive subgraph as $SSubG = \{ssubg_1, ssubg_2, \ldots, ssubg_n\}$. For each sensitive API node api_i corrordings a sensitive subgraph as $ssubg_i$, which involves calling other functions that can be other sensitive API nodes or not. We define a distance function $\mathrm{dis}(api_i, v_k)$, which finds the shortest execution path between the sensitive API node api_i and its k-hop neighbor v_k, where k is used to control the size of a sensitive subgraph $ssubg_i$. For each sensitive subgraph $ssubg_i$, we also calculate its maliciousness degree as follows.

$$m\left(ssubg_i\right) = \sum_{api_i \in M_{api}(ssubg_i)} m\left(api_i\right) \tag{4}$$

Intuitively, the importance of each sensitive subgraph $ssubg_i$ is different for benign and malicious software. Therefore, assigning the same weight to each sensitive subgraph $ssubg_i$ may make it difficult to distinguish benign and malicious apps, affecting the malware detection accuracy. To solve this problem, we propose an adaptive weight allocation method to adaptively select a weight to measure the malicious degree of each sensitive subgraph $ssubg_i$ as follows.

$$I\left(ssubg_i\right) = \max\left\{1, \frac{|ssubg_i| * n}{\sum_{j=1}^{n} |ssubg_j|}\right\}$$

where $|ssubg_i|$ is the size of a sensitive subgraph $ssubg_i$, and n is the size of the set of sensitive subgraph $SSubG$. Finally, we can extract a sensitive subgraph feature set as follows.

$$\begin{aligned} F_{SSubG} = & \{[I\left(ssubg_1\right) \cdot m\left(ssubg_1\right)], \\ & [I\left(ssubg_2\right) \cdot m\left(ssubg_2\right)], \\ & \ldots, [I\left(ssubg_n\right) \cdot \mathrm{m}\left(ssubg_n\right)]\} \\ = & \{f_{s_1}, f_{s_2}, \ldots, f_{s_n}\}. \end{aligned} \tag{5}$$

2) Semantic Vector Generation: The extracted the sensitive subgraph features in the previous process are stored in a text database. In order to model higher-level concepts and facts in malware, we tried to extract the potential semantic patterns from this text database. In our case, we use the word embedding method to inject the semantic feature into a embedding vector space, which reflects the latent behavioral patterns of malware. In this vector space, the semantic vector, denoted by V_{sem}, is represented by a set of hidden variables, and each sensitive API node api_i from the sensitive subgraph $SSubG$ is represented by a specic instantiation of these variables. This method is spatially insensitive, regardless of the order of words or local patterns. By doing this, the semantic feature verctors for each malware maps the semantic-knowledge into a fixed two-dimensional matrix $E^{M,K}$, where M and K represent the maximum number of semantic features and the embedding size of vectors, respectively. Finally, we transform sensitive subgraph features F_{SSubG} into semantic features as follows:

$$F = \{f_1, f_2, \ldots, f_n\}. \tag{6}$$

After that, the semantic features F are fed into the detection model as an input. Those semantic patterns are aggregated at lower levels, which promotes the representation of higher-level domain knowledge, and enables us to better identify patterns of features.

3) Frequency Statistics Analysis: To investigate the effectiveness of the semantic features, we analyze some relevant statistics based on the semantic features. Fig. 4 shows the frequency statistics of the semantic features on the AZ dataset. Most semantic features appear only a few times, which may introduce additional bias into the evaluation. It will create sparser context

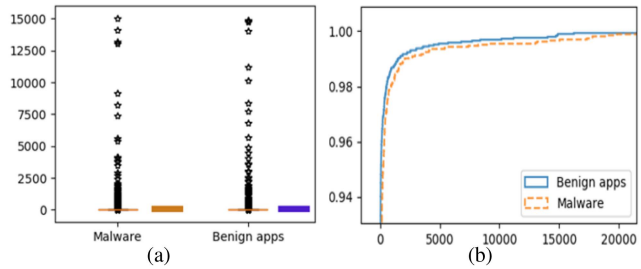


Fig. 4. Frequency statistics.

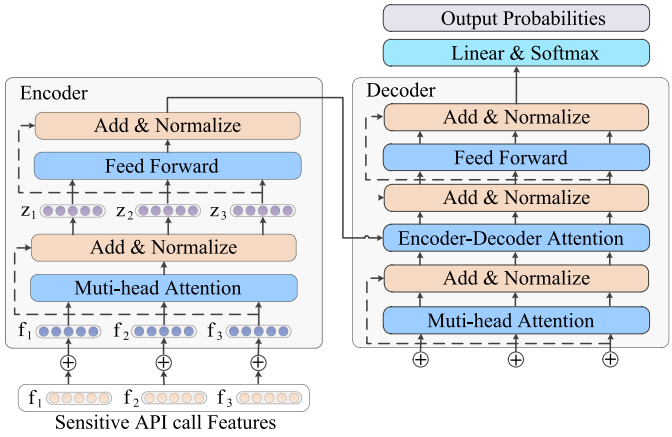


Fig. 5. The proposed model architecture.

descriptions and words that share contexts are not enough in the embedded vector space. Thus, it is hard for a learning model to train high-quality embeddings to capture sequential patterns due to the diversity and low-frequency of those semantic expressions. This effect can be countered by ignoring the low frequency features when calculating the embedding vector of each semantic feature. This strategy enables our model to refine contextual information and train on a larger dataset. In fact, some specific API calls are frequently used. This is important for revealing malicious behavior patterns, and provides effective information gain for model learning since the differences in feature frequency distribution can reflect the inconsistencies between malware and benign apps.

C. Model Training and Updating on Cloud

It is clear that the cloud farther from the end users has high computing power. The large-scale deep learning models are more suitable to be created and trained in the cloud to help the threat detection. Specifically, the detection model will be retrained periodically to update its detection capabilities.

In this article, we first extract the sensitive subgraph features from a malware. Then, we build the Transformer-based malware detection model to capture remote context dependencies. Transformer is an attention-based sequence-to-sequence model. Since each sensitive subgraph feature contributes differently to malware detection, the Transformer model uses attention to focus on the key sensitive subgraph features that are important to the semantic representation. In our case, the attention acts somewhat as an optimized feature extractor on the sequences of features. In addition, considering that the lack of location information will lead to the confusion of contextual semantic information of the output sequence, it is difficult for the learning model to train high-quality embedding to capture sequential patterns. To address this, the Transformer model uses location embedding as an input to improve the modeling ability of the learning model to integrate forecast knowledge. The overall flow of the proposed algorithm is shown in Fig. 5 and Algorithm 2.

For more details, given the linear projections Q, K, V, the attention used in Transformer maps them to an output, which can be treated as a compatibility function that calculates the probability distribution of attention. In our case, the attention computes the attention weights by the dot products of the query Q with the corresponding keys K. Then, we use a softmax function to compute the weights on the values. This process

Algorithm 2: Learning Model on the Cloud.

Input: The semantic embedding vectors $F = \{f_1, f_2, \ldots, f_n\}$ extracted from the sensitive subgraph sequence F_{SSubG};
Output: The optimized learning model;
Initialize all the model parameters;
Feed the semantic embedding vectors $F = \{f_1, f_2, \ldots, f_n\}$ into the learning model;
while *not converged* **do**
Optimize learning model using (7)–(10);
Update model parameters.
end while
return The optimized learning model.

can be described as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{7}$$

where d_k is the dimension of the input queries Q and keys K, and $\frac{1}{\sqrt{d_k}}$ is a scaling factor.

Moreover, we use the multi-head attention to attend to information from different representation subspaces at different positions. Compare to the single attention, the multi-head attention performs the attention function in parallel, which projects Q, K and V h times with different linear projections to d_k, d_k and d_v dimensions, respectively. In this case, each attention yields a output vector. Finally, the outputs from h attention function are concatenated together, which then yields a final output by the linear layer with Softmax function. Multi-head attention enables our model to capture important features in different subword spaces, where the sensitive API features in different spaces contribute differently to the final representation. Fig. 6 give the details of multi-head attention mechanism. This process can be described as follows.

$$\text{MultiHead}(Q, K, V) = Concat\left(head_1, \ldots, head_h\right)W^O, \tag{8}$$

$$head_i = \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right), \tag{9}$$

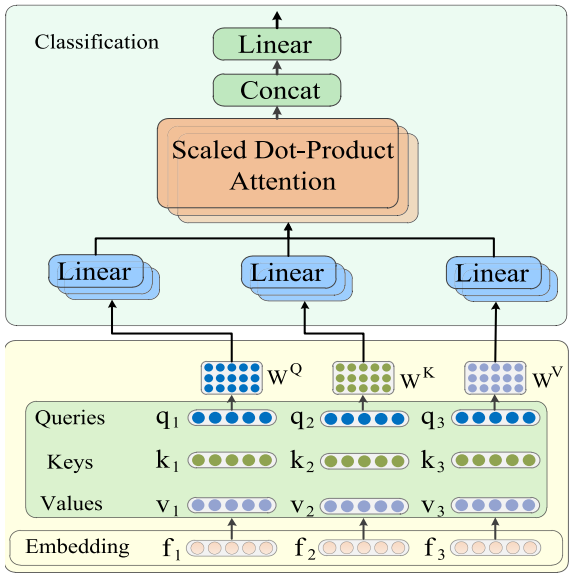


Fig. 6. Multi-Head attention mechanism used in our model, which consists of several attention layers running in parallel.

where $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$. are the parameter matrices of linear transformations. In this article, we set the number h of heads (i.e., the attention function in the multi-head attention) as 8. Specifically, we employ residual connection and normalization techniques in the multi-head attention. The residual connection can pass the local representation of the lower layer to the upper layer to obtain a higher-level local representation. Furthermore, we adopt a fully connected feed-forward network FFN. Then, the output vectors from the multi-head attention layer are fed into the FFN layer to obtained the hidden representations. It can be described as:

$$\text{FFN}(x) = \max\left(0, xW_1 + b_1\right) W_2 + b_2, \tag{10}$$

where W_1 and W_2 are the weight matrices. b_1 and b_2 are the biases.

V. Experiments and Evaluation

In IoT network, IoT devices typically have multiple applications installed. Assuming that the purpose of the adversaries is to install their malwares on the target Mobile-IoT devices whlie spreading the malwares for infecting more devices. Therefore, we should carefully check these applications to ensure the security of IoT devices. In the following section, we give the details of the experimental setup for evaluating the proposed TransMalDE in malware detection and conduct a large number of experiments to evaluate the effectiveness of TransMalDE.

Using the Tensorflow and Keras platforms, we implement the Transformer-based malware detection model, and use word2vec to extract the semantic embeddings. The evaluation shows that our method can detect malware effectively.

TABLE II
Main Datasets Used in Our Evaluation Studies

	Benign apps		Malware		
Datasets	Source	#Apps	Source	#Apps	#Total
DR	GP	5560	DR	5560	11,120
MD	GP	1,0320	MD	1,0320	20,640
VS	GP	1,1497	VS	1,1497	22,994
AZ	AZ	15,278	AZ	15,278	30,556

A. Datasets

The availability of representative data is a key challenge in malware detection. We used four publicly available data sets to evaluate TransMalDE's performance. Drebin (DR) [30] contains 5560 malware samples collected from August 2010 to October 2012, each labeled as one of 179 malware families, such as DroidDream, FakeDoc, Geinimi, etc. All labels are created after being scanned by different Anti-Virus (AV) scanners. MalDroid (MD) [32] is the latest Android malware dataset collected from December 2017 to December 2018. This dataset is intentionally spanning five distinct categories: Adware, Banking malware, SMS malware, Riskware, and Benign. In our experiment, we randomly selected 10,320 malware samples. VirusShare [33] is an online site, and we collected 11,497 malware samples from VirusShare to build the VS dataset. AndroZoo (AZ) [34] dataset contains all published applications from 2010 to the present, including benign apps and malware. Specifically, each malware in this dataset has been analyzed by tens of different AV scanners.

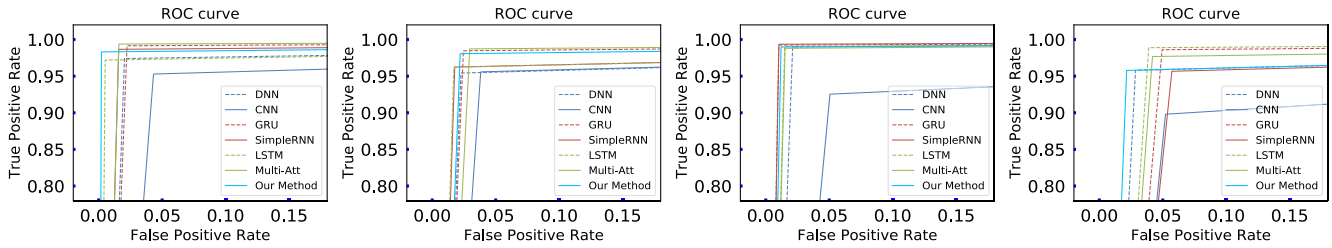
It is worth mentioning that the previous detection system has the problem of class imbalance during the model training. However, the detection model trained on imbalanced data is not enough to truly reflect performance to measure whether the detection model provides results that are statistically generalizable. To make a fair comparison, we set the same number of malware and benign applications during training to prevent the dataset from being highly unbalanced. Therefore, to be fair, we collected benign samples from the Google App Store to balance out the datasets. This means that the number of benign and malicious samples is equal in our experiment. Table II provides a summary of the various datasets used in this article.

B. Evaluation Metrics

In our experiment, we used some commonly used machine learning performance evaluation metrics to evaluate the classification results of the TransMalDE model, including accuracy, precision, recall and F-score (F). In addition, we also use Receiver Operating Characteristic (ROC) curves to evaluate the performance of our model in malware detection. Our goal is to achieve high accuracy and F1 values.

C. Comparative Classification Performance

To highlight the significance of this study, we compared TransMalDE to a number of shallow machine learning models, including Linear regression (LR), Random Forest (RF) and support vector machine (SVM). Evaluation results are shown in Table III. It is clear that our TransMalDE achieved better



(a) Comparisons on DR dataset (b) Comparisons on VR dataset (c) Comparisons on MD dataset (d) Comparisons on AZ dataset

Fig. 7. TransMalDE versus baselines for malware detection on four benchmark datasets.

TABLE III
TRANSMALDE VERSUS SHALLOW MACHINE LEARNING MODELS FOR MALWARE DETECTION ON FOUR BENCHMARK DATASETS

Datasets		LR	SVM	RF	Our Method
DR	Acc	65.01	88.69	97.53	99.06
	P	61.71	87.95	98.31	99.76
	R	75.94	88.69	96.67	98.31
	F	68.09	88.32	97.48	99.03
VR	Acc	87.86	95.32	99.07	97.92
	P	84.81	94.89	98.45	97.37
	R	88.58	94.74	99.49	98.07
	F	86.66	94.82	98.97	97.72
MD	Acc	71.94	90.61	98.39	98.97
	P	69.78	92.26	98.07	98.52
	R	83.28	89.66	98.94	99.59
	F	75.94	90.94	98.50	99.05
AZ	Acc	72.16	97.24	98.88	96.98
	P	74.22	92.26	98.94	96.92
	R	47.87	89.66	98.28	95.78
	F	58.20	90.94	98.61	96.35

performance than the LR and SVM models on the four benchmark data sets. Compared with these two models, RF achieves a relatively better result. Moreover, we compared the proposed TransMalDE model with previous state-of-the-art models, such as CNN [35], LSTM [36], and DNN [37]. Table IV and Fig. 7 show the evaluation results. It can be observed that CNN has the worst performance on the four benchmark datasets, which may be because CNN is more used in the field of computer vision and it is difficult to extract beneficial information from semantic features. Moreover, we can see that Muti-Att is closest to our TransMalDE model in F-score on the four benchmark datasets. Compared with the proposed TransMalDE model, Muti-Att uses only multi-head attention to learn the semantic information, which models the hidden contextual by calculating the attention weights. The excellent performance of Muti-Att on four benchmark datasets verifies the effectiveness and necessity of multi-head attention, which can reveal malicious behavior patterns.

Moreover, the baseline LSTM has achieved good performance, which utilizes the hidden contextual representation to model the semantic information. GRU and SimpleRNN achieve the best performance on the MD dataset, even exceeding our proposed TransMalDE model, with an improvement of 0.1% and 0.14%, respectively. This may be because they take into account the contextual information in the processing sequence, which can retain a lot of local information. This result shows the effectiveness and powerful generalization of GRU and SRNN in detecting malware. However, GRU and SimpleRNN perform less competitive than the proposed TransMalDE model on the other three benchmark datasets. Overall, the proposed TransMalDE produces the best performance under all metrics, which helps the model to obtain more information related to malware behaviors. This demonstrates the superiority of our model proposed in this article.

D. Length of Sensitive Subgraph Sequence

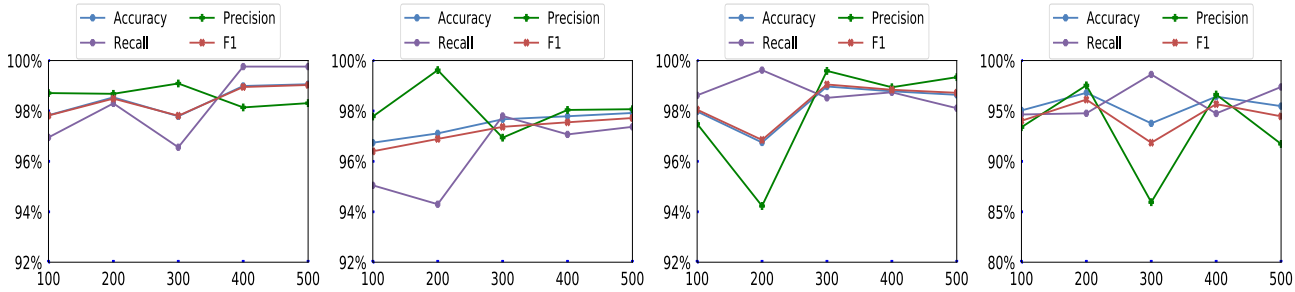
The maximum length of sensitive subgraph sequence is an important parameter in constructing semantic vector, which directly affects the amount of the semantic information contained in the semantic vectors. Fig. 8 shows the performance of the model's evaluation index when the maximum sequence length is 200. It is clear that the best value of length is 500 for both DR And VR datasets. With the larger value of length, the more semantic information is contained in the sensitive subgraph sequence. For MD and AZ datasets, the best value of length is 300 and 200, respectively. Although the amount of semantic information becomes larger, performance changes over time. However, semantic information that is too far away has little effect on the current learning of the model.

E. Dimension of Semantic Vector

The dimension of semantic vectors plays an important role in the TransMalDE model. We explore the effect of the dimensionality of semantic vectors on model performance. Specifically, we change the dimension of the semantic vector in the range [30, 50, 100, 150, 200]. Fig. 9 summarizes the performance of the TransMalDE model in the dimensions of different semantic vectors. For DR, VR and AZ datasets, the TransMalDE model achieves the best performance when the dimension of the semantic vector is set to 100. As the dimension of the semantic vector increases, the more semantic information it contains, the better the performance of the TransMalDE model gradually becomes. In this case, sufficient semantic information can enhance the representation ability of the TransMalDE model. In addition, for MD dataset, the TransMalDE model achieves the best performance when the dimension of semantic vectors is small. However, as the semantic dimension is gradually increased, we find that the performance of the TransMalDE model degrades. This indicates that too much semantic information will introduce noise to the model, resulting in poor model performance. Specifically, the changing trend shows that our TransMalDE always maintain a high F-score on four benchmark datasets.

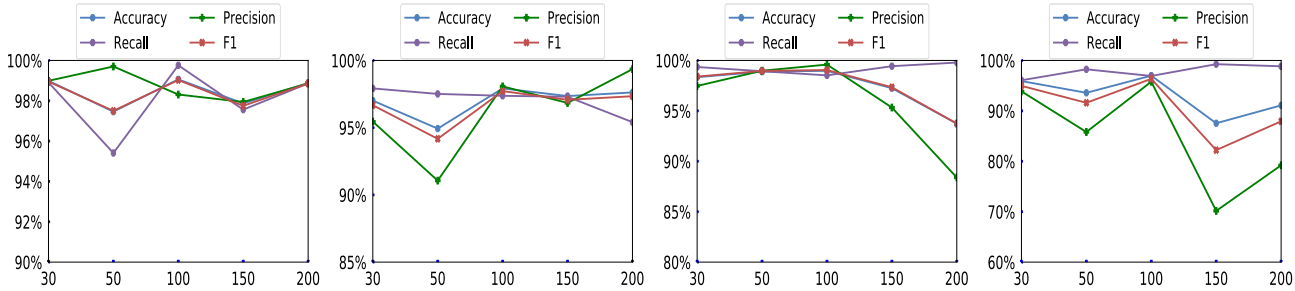
TABLE IV
TRANSMALDE VERSUS BASELINES FOR MALWARE DETECTION ON FOUR BENCHMARK DATASETS

Datasets		DNN	CNN	GRU	SimpleRNN	LSTM	Multi-Att	Our Method
DR	Acc	97.50	95.48	98.43	98.54	98.35	98.88	99.06
	P	98.98	95.50	97.64	99.21	99.45	98.32	99.76
	R	95.89	95.28	99.14	97.81	97.20	99.38	98.31
	F	97.41	95.39	98.38	98.50	98.31	98.85	99.03
VR	Acc	96.70	95.91	97.95	97.33	97.37	97.79	97.92
	P	97.11	95.39	97.00	97.78	97.84	96.45	97.37
	R	95.42	95.58	98.46	96.24	96.21	98.72	98.07
	F	96.26	95.49	97.72	97.00	97.02	97.57	97.72
MD	Acc	98.67	93.66	99.11	99.16	98.78	98.64	98.97
	P	98.04	95.37	99.06	99.02	98.65	98.62	98.52
	R	99.49	92.53	99.24	99.35	99.03	98.80	99.59
	F	98.76	93.93	99.15	99.19	98.84	98.71	99.05
AZ	Acc	96.08	92.72	96.49	94.84	96.57	96.07	96.98
	P	93.15	92.41	93.19	92.05	94.17	94.45	96.92
	R	97.53	89.80	98.60	95.66	97.68	96.17	95.78
	F	95.29	91.09	95.82	93.82	95.89	95.20	96.35



(a) Comparisons on DR dataset (b) Comparisons on VR dataset (c) Comparisons on MD dataset (d) Comparisons on AZ dataset

Fig. 8. Comparative classification performance in various feature numbers.



(a) Comparisons on DR dataset (b) Comparisons on VR dataset (c) Comparisons on MD dataset (d) Comparisons on AZ dataset

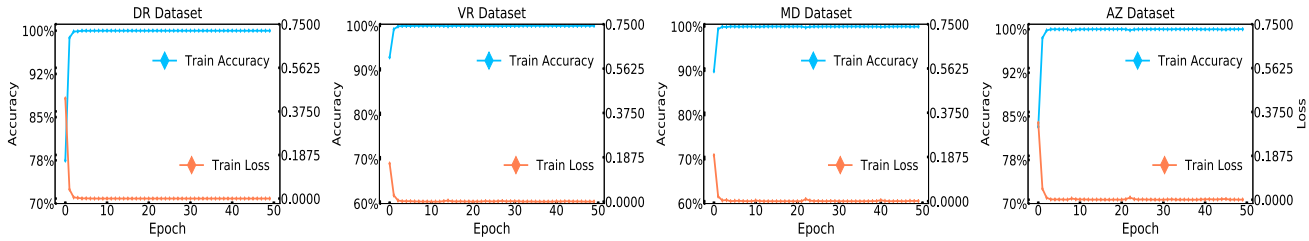
Fig. 9. Comparative classification performance in various semantic vector dimensions.

F. Efficiency of TransMalDE

To illustrate our training progress, we plotted performances of TransMalDE. In our case, TransMalDE is trained for 50 epochs, and then its performance saturates at a certain point. Fig. 10 shows the evolution of the training loss and accuracy of our TransMalDE over 50 epochs. We can see that the training loss and accuracy of our TransMalDE on the four benchmark datasets are similar. The model can converge to the similar validation accuracy in all four benchmark datasets. The learning curves on the four datasets ascend to a plateau quickly. This means that our TransMalDE can converge quickly in the training process. Overall, the results indicate that the proposed TransMalDE has a beneficial performance on malware detection.

G. Robustness

To evaluate the resiliency of TransMalDE for sophisticated obfuscation schemes, we used four obfuscation datasets consisting of samples from the AZ and PG datasets. The percentage of obfuscated malicious samples denoted by obf in Fig. 11. As we expect, even malicious samples were increasingly obfuscated, TransMalDE can still achieve good performance. The ROC results corroborated the robustness merits of TransMalDE. This is because the embedding operations count the frequency of



(a) Comparisons on DR dataset (b) Comparisons on VR dataset (c) Comparisons on MD dataset (d) Comparisons on AZ dataset

Fig. 10. Evolution of the training loss and accuracy of our TransMalDE during training on four benchmark datasets.

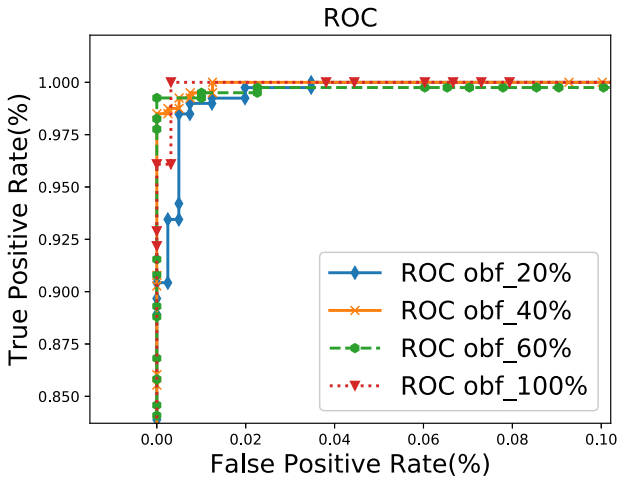


Fig. 11. Classification evaluation of different obfuscated datasets.

occurrence of words directly from the source texts and the attention focus on words with higher frequency. This approach designed mitigates the effects of obfuscation techniques that introduce a large number of useless system calls.

In addition, malware developers also often confuse their malware by renaming properties (such as classes, fields, and methods). We also can effectively reduce the impacts of such confusing schemes to some extent by focusing on behavioral information (e.g. sensitive and confusing-related API calls). Overall, the ROC results show that TransMalDE performs consistently well on all datasets, even in the presence of complicated obfuscation. Another interesting find is that, despite various obfuscated schemes used, we achieved similar results to what were reported by DroidCat [38] and MamaDroid [39]. Thus, our approach is comparable to the two state-of-the-art approaches [38], [39].

VI. Conclusion

Edge computing paradigm provides a greater quality of service (QoS) by massive Internet of Things (IoT) applications, which enables computing services at the edge of network. However, this raises the potential risk of being attacked by malware. Existing malware detection systems based on deep learning methods are difficult to deploy battery-powered iot devices because they are limited by resources and computation. To address this challenge, we designed a hierarchical security architecture to detect malware, where users computationally intensive tasks migrate to adjacent edge computing nodes. The main advantage of TransMalDE is the high execution efficiency of malware detection against the rival malware detection systems. In the future research, we will try to build a more effective hierarchical architecture to further help improve the operation efficiency of the malware detection system.

In this article, we used the static analysis method to extract sensitive subgraph features, which provides useful information for subsequent model learning. Dynamic analysis can provide a more thorough malware detection mechanism, and effectively reduce false positives, but the computational cost is much higher than that of static analysis. Future research will consider combining static and dynamic analysis methods to improve malware detection performance. In addition, we plan to improve the proposed Transformer based malware detection model, such as replacing components in the Transformer model with newer, better performing versions.

References

[1] J. Jeon, B. Jeong, S. Baek, and Y. 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.

[2] 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 Secur. Comput.*, vol. 20, no. 3, pp. 2127–2143, May/Jun. 2023.

[3] S. Liu, J. Yu, X. Deng, and S. Wan, "FedCPF: An efficient-communication federated learning approach for vehicular edge computing in 6G communication networks," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 2, pp. 1616–1629, Feb. 2022.

[4] 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.

[5] K. Vahedi and K. Afhamisisi, "Cloud based malware detection through behavioral entropy," in *Proc. IEEE Int. Conf. Big Data*, Orlando, FL, USA, 2021. pp. 6046–6048.

[6] J. C. Kimmell, M. Abdelsalam, and M. Gupta, "Analyzing machine learning approaches for online malware detection in cloud," in *Proc. IEEE Int. Conf. Smart Comput.*, Irvine, CA, USA, 2021. pp. 189–196.

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

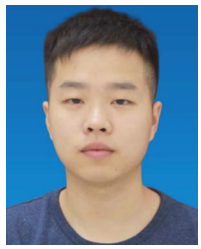
[8] D. Nahmias, A. Cohen, N. Nissim, and Y. Elovici, "Deep feature transfer learning for trusted and automated malware signature generation in private cloud environments," *Neural Netw.*, vol. 124, pp. 243–257, 2020.

[9] 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.

[10] 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.

[11] C. Li et al., "DMalNet: Dynamic malware analysis based on API feature engineering and graph learning," *Comput. Secur.*, vol. 122, 2022, Art. no. 102872.
[12] Z. Zhang, Y. Li, H. Dong, H. Gao, Y. Jin, and W. Wang, "Spectral-based directed graph network for malware detection," *IEEE Trans. Netw. Sci. Eng.*, vol. 8, no. 2, pp. 957–970, Apr.-Jun. 2021.
[13] A. Azmoodeh, A. Dehghantanha, and K.-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.-Mar. 2019.
[14] Y. Qiao, W. Zhang, X. Du, and M. Guizani, "Malware classification based on multilayer perception and Word2Vec for IoT security," *ACM Trans. Internet Techn.*, vol. 22, no. 1, 2022, Art. no. 10.
[15] M. Amin, B. Shah, A. Sharif, T. Ali, K. Kim, and S. Anwar, "Android malware detection through generative adversarial networks," *Trans. Emerg. Telecommun. Technol.*, vol. 33, no. 2, 2022, Art. no. e3675, doi: 10.1002/ett.3675.
[16] M. Amin, T. A. Tanveer, M. Tehseen, M. Khan, F. A. Khan, and S. Anwar, "Static malware detection and attribution in Android byte-code through an end-to-end deep system," *Future Gener. Comput. Syst.*, vol. 102, pp. 112–126, 2020, doi: 10.1016/j.future.2019.07.070.
[17] M. Amin, D. Shehwar, A. Ullah, T. Guarda, T. A. Tanveer, and S. Anwar, "A deep learning system for health care IoT and smartphone malware detection," *Neural Comput. Appl.*, vol. 34, no. 14, pp. 11283–11294, 2022, doi: 10.1007/s00521-020-05429-x.
[18] A. Ferdowsi and W. Saad, "Deep learning for signal authentication and security in massive internet-of-things systems," *IEEE Trans. Commun.*, vol. 67, no. 2, pp. 1371–1387, Feb. 2019.
[19] Y. Liu, X. Ma, J. Bailey, and F. Lu, "Reflection backdoor: A natural backdoor attack on deep neural networks," in *Proc. 16th Eur. Conf. Comput. Vis.*, 2020, pp. 182–199, doi: 10.1007/978-3-030-58607-2_11.
[20] 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. Serv. Manag.*, vol. 20, no. 1, pp. 684–696, Mar. 2023.
[21] E. Quiring and K. Rieck, "Backdooring and poisoning neural networks with image-scaling attacks," in *Proc. IEEE Secur. Privacy Workshops*, San Francisco, CA, USA, 2020. pp. 41–47, doi: 10.1109/SPW50608.2020.00024.
[22] S. Rasthofer, S. Arzt, S. Triller, and M. Pradel, "Making malory behave maliciously: Targeted fuzzing of Android execution environments," in *Proc. IEEE/ACM 39th Int. Conf. Softw. Eng.*, 2017, pp. 300–311.
[23] J. Fairbanks, A. Orbe, C. Patterson, E. Serra, and M. Scheepers, "Identifying att&ck tactics in Android malware control flow graph through graph representation learning and interpretability (student abstract)," in *Proc. 36th AAAI Conf. Artif. Intell.*, 2022, pp. 12941–12942.
[24] C. Wu, T. Ban, S. Cheng, T. Takahashi, and D. Inoue, "Iot malware classification based on reinterpreted function-call graphs," *Comput. Secur.*, vol. 125, 2023, Art. no. 103060.
[25] J. Jia and P. K. Chan, "Representation learning with function call graph transformations for malware open set recognition," in *Proc. IEEE Int. Joint Conf. Neural Netw.*, Padua, Italy, 2022, pp. 1–8.
[26] W. Yuan, Y. Jiang, H. Li, and M. Cai, "A lightweight on-device detection method for Android malware," *IEEE Trans. Syst. Man Cybern. Syst.*, vol. 51, no. 9, pp. 5600–5611, Sep. 2021.
[27] R. Feng, S. Chen, X. Xie, G. Meng, S. Lin, and Y. Liu, "A performance-sensitive malware detection system using deep learning on mobile devices," *IEEE Trans. Inf. Forensics Secur.*, vol. 16, pp. 1563–1578, 2021.
[28] W. Niu, X. Zhang, X. Du, T. Hu, X. Xie, and N. Guizani, "Detecting malware on X86-based IoT devices in autonomous driving," *IEEE Wirel. Commun.*, vol. 26, no. 4, pp. 80–87, Aug. 2019.
[29] 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.
[30] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck, "Drebin: Effective and explainable detection of Android malware in your pocket," in *Netw. Distrib. Syst. Secur. Symp.*, 2014, pp. 23–26.
[31] M. Fan et al., "Android malware familial classification and representative sample selection via frequent subgraph analysis," *IEEE Trans. Inf. Forensics Securi.*, vol. 13, no. 8, pp. 1890–1905, Aug. 2018.
[32] 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.
[33] "Virusshare," 2021. [Online]. Available: http://virusshare.com/
[34] 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.
[35] N. McLaughlin et al., "Deep Android malware detection," in *Proc. 7th ACM Conf. Data Appl. Secur. Privacy*, 2017, pp. 301–308.
[36] Z.-U. Rehman et al., "Machine learning-assisted signature and heuristic-based detection of malwares in Android devices," *Comput. Elect. Eng.*, vol. 69, pp. 828–841, 2017.
[37] 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.
[38] H. Cai, N. Meng, B. Ryder, and D. Yao, "DroidCat: Effective Android malware detection and categorization via app-level profiling," *IEEE Trans. Inf. Forensics Secur.*, vol. 14, no. 6, pp. 1455–1470, Jun. 2019.
[39] L. Onwuzurike et al., "Mamadroid: Detecting Android malware by building Markov chains of behavioral models (extended version)," *ACM Trans. Privacy Secur. (TOPS) Arch.*, vol. 22, no. 2, 2019, Art. no. 14.

Xiaoheng Deng (Senior Member, IEEE) received the Ph.D. degrees 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, Changsha, China. He is a Joint Researcher of Shenzhen Research Institue, Central South University. His research interests include network security, edge computing, Internet of Things, online social network analysis, data mining, and pattern recognization. He is a Senior Member of CCF, a member of CCF Pervasive Computing Council, a member of ACM. He was the Chair of CCF YOCSEF CHANGSHA from 2009 to 2010.



Zhe Wang is currently working toward the M.S. degree from the School of Computer Science and Engineering, Central South University, Changsha, China. Since 2020, he has been engaged in the direction of cyber security. His research interests include data minig, malware detection, edge computing, and IoT security.

Xinjun Pei (Student Member) is currently working toward 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 IoT security.

Kaiping Xue (Senior Member, IEEE) received the bachelor's degree from the Department of Information Security, University of Science and Technology of China (USTC), Hefei, China, in 2003, and the Ph.D. degree from the Department of Electronic Engineering and Information Science, 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, Gainesville, FL, USA. He is currently 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), the IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS (TWC), and the IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT (TNSM). He has also was the (Lead) Guest Editor for many reputed journals/magazines, including IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS (JSAC), *IEEE Communications Magazine*, and IEEE NETWORK. He is an IET Fellow.