Computers & Security 140 (2024) 103752

Contents lists available at ScienceDirect

Computers & Security

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

TS-Mal: Malware detection model using temporal and structural features learning

Wanyu Li [a], Hailiang Tang [b], Hailin Zhu [b,*], Wenxiao Zhang [c], Chen Liu [d]

[a] School of Literature, History and Culture, Qilu Normal University, Jinan, China
[b] School of Information Science and Engineering, Qilu Normal University, Jinan, China
[c] School of Finance and Economics, Shandong University of Engineering and Vocational Technology, Jinan, China
[d] School of Computer Science and Engineering, Beihang University, Beijing, China

ARTICLE INFO

Keywords:
Malware detection
Graph attention network
API call events
Temporal feature

ABSTRACT

The cyber ecosystem is facing severe threats from malware attacks, making it imperative to detect malware to safeguard a purified Internet environment. However, current studies primarily concentrate on examining the time-based correlation between APIs for malware detection while neglecting the contextual associations derived from API categories, resulting in inadequate detection performance. In this paper, we present TS-Mal, a novel Malware detection model incorporated Temporal and Structural features learning. Particularly, TS-Mal first designs a temporal vector learning method to automatically capture the evolving representation from the non-repetitive API sequences, which can efficiently pursue the attack preferences of malware. Then TS-Mal introduces heterogeneous graphs to model the interactive relationships between APIs and presents a dense-interactive structural embedding approach to generate the fine-grained API structural representation, which is capable of utilizing API category interaction information to boost detection effectiveness. Finally, TS-Mal simultaneously integrates temporal and structural attack features to accurately identify the unknown malware, effectively defending against new malware attacks. Experimental results on real-world datasets demonstrate that our proposed TS-Mal model outperforms existing state-of-the-art methods.

1. Introduction

In recent years, the mobile application market has been plagued by an increasing number of malware attacks. The proliferation of malware variant transformation technology has led to an increase in both the quantity and quality of various malware, consequently elevating the level of threat (Chen et al., 2019). More specifically, malware transformation techniques, also known as malware evasion techniques, involve malware producers seeking new means of infection, making initial versions of the malware "brand new" with only minor modifications, thereby evading security controls. At present, the popular variant transformation techniques include packaging, encryption, code obfuscation, direct system calls, and so on. Obviously, since variant transformation technology is a modification of the original malware samples, it takes much less time and technical cost than creating entirely new malware, resulting in the proliferation of malware variants. According to a recent report, Sophos predicts that the incidence of malware attacks will increase by 29% from 2021 to 2023 (Sophos, 2022). Thus, detecting malware has been an urgent and important task for mitigating cyber threats.

In the past few years, most efforts on malware analysis (Pascanu et al., 2015; Sun et al., 2016; Kang et al., 2016; Gaviria de la Puerta and Sanz, 2017; Raff et al., 2018; Chen et al., 2019; Zhang et al., 2019, 2020; Abdelnabi et al., 2020; Wang et al., 2020; Liu et al., 2021) focused on detection or classification (identifying malware and their variants from benign software). Generally, the existing malware detection approaches can be roughly divided into static-based methods (Kang et al., 2016; Gaviria de la Puerta and Sanz, 2017; Raff et al., 2018; Zhang et al., 2019) and dynamic-based methods (Pascanu et al., 2015; Sun et al., 2016; Zhang et al., 2018; Chen et al., 2019; Zhang et al., 2020; Abdelnabi et al., 2020; Liu et al., 2021, 2022, 2023b). More specifically, the static-based approaches aim to leverage machine learning or deep learning for extracting malware fingerprints from large-scale known malware samples to identify potential malware samples. Unfortunately, the efficacy of these static-based approaches is compromised as new malware variants, generated through code obfuscation or packing tech-

* Corresponding author.
E-mail addresses: 15854180098@163.com (W. Li), China.zhuhl_qlnu@163.com (H. Zhu).

https://doi.org/10.1016/j.cose.2024.103752
Received 16 October 2023; Received in revised form 29 December 2023; Accepted 5 February 2024
Available online 12 February 2024

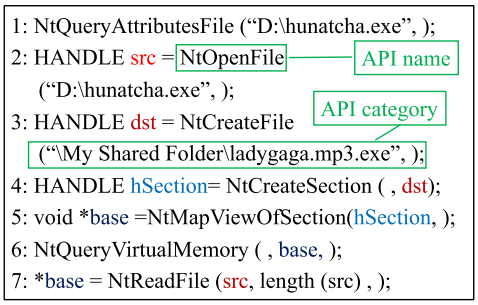


Fig. 1. An illustration of API call events, which consists of API call sequences (i.e., API names interaction) and API behavioral interdependence (i.e., API categories interaction).

niques, can effortlessly evade detection by tampering with their static signatures (Pascanu et al., 2015). The dynamic-based approaches provide a promising solution for detecting malware variants utilizing code obfuscation or packing techniques (Zhang et al., 2020). They mainly concentrate on extracting temporal features from API sequences and applying deep neural networks to capture the dynamic evolution patterns of malware. Nevertheless, it can be observed that malware often hides malicious APIs in a large number of benign APIs to evade detection (Liu et al., 2023a). This results in high false positives for API-based analysis methods that ignore the meaningful contextual interactions from API parameters (Kawakoya et al., 2019).

In fact, the majority of newly emerged malware has evolved from known malware families. Although tremendous research efforts have been made, there are still two major limitations in the existing static analysis and dynamic analysis techniques that hinder them from being applied to accurately and robustly detect malware. Firstly, they mostly extract isolated features to represent executable files, ignoring the fact that malware contains both temporal evolution and semantic information. Secondly, the existing dynamic analysis techniques mainly adopt traditional RNN to extract the temporal dependence from the original API sequence and introduce a lot of noise from repeated behaviors, so they cannot provide effective detection for delayed malware variants. To address the aforementioned limitations, we argue that discriminatively dividing the original malware execution behavior into fine-grained API name sequences and underlying API interactions is capable of characterizing the malware attack pattern from different perspectives, and the pre-processed API behavior is immune to delayed attacks. However, the current challenges of designing an accurate and robust malware detection technology are as follows:

- **Challenge 1:** The execution behavior of malware is composed of a sequence of API call events, shown in Fig. 1. These events contain temporal API call information (i.e., API name interaction) and underlying behavioral interdependence (i.e., API category interaction). Furthermore, malware frequently leverages obfuscation techniques to hide malicious behavior within extensive benign activities (Wang and Philip, 2019). Drawing upon these observations, integrating API temporal evolutionary patterns with API contextual structural patterns to improve detection capability is fundamentally challenging.
- **Challenge 2:** A conceivable challenge is how to efficiently model historical API name call sequences and employ their temporal features to infer malware's attack intents rather than manually simulating all possible paths based on malware's historical execution behavior, which incurs undesirable time costs.
- **Challenge 3:** Existing heterogeneous graph learning focus on straightforwardly aggregating the meta graph-based explicit neighborhoods into the single node embedding (Qu et al., 2019; Li et al., 2022), which is sparse and partial; hence, how to mine richer and more valuable implicit neighbors to improve the malware detection capability is exceedingly challenging.

To combat the aforementioned challenges, this paper presents TS-Mal, a novel malware detection model integrating temporal and structural features learning to accurately detect unknown malware. In particular, the former studies fine-grained API name sequences to capture temporal dependencies between different API operations, while the latter studies underlying API interaction information to learn interaction semantics between each system entity. The main contributions of this work can be summarized as follows:

- **Contribution 1:** We present TS-Mal to detect malware based on multi-dimensional API call events. TS-Mal examines the API execution events of malware to investigate their temporal API call sequences and underlying API category interactions, which can simultaneously capture comprehensive malware features from temporal and structural perspectives.
- **Contribution 2:** We propose a temporal vector learning method to effectively capture the evolutionary intents of attacks by analyzing temporal sequences of API calls. Additionally, we introduce an API subsequences filtering algorithm to reduce training costs, ensuring non-repetitive API name sequences that preserve critical attack information and enhance real-time detection efficiency.
- **Contribution 3:** The malware API category interactions are modeled using heterogeneous graphs, and a dense-interactive graph embedding method is proposed to generate a fine-grained structural representation. This method has the ability to investigate comprehensive contextual semantic knowledge from the explicit and implicit neighbors to enhance the effectiveness of malware detection.
- **Contribution 4:** We comprehensively evaluate the effectiveness and efficiency of TS-Mal on three real-world malware datasets. Experimental results verify that our proposed TS-Mal outperforms state-of-the-art approaches in detecting newly emerged malware attacks.

The remaining sections of this paper are organized as follows. Section 2 reviews the related works. Section 3 provides the preliminaries used throughout this study. Section 4 illustrates the details of our proposed TS-Mal. In section 5, we conduct extensive experimental analysis to evaluate the performance of TS-Mal. Finally, the conclusion is summarized in Section 6.

2. Related works

Many research studies have employed machine learning techniques or deep learning techniques for malware detection and classification. These works generally take static, dynamic, or heterogeneous graphs as input and output binary class (e.g., benign or malicious) or multi-class (e.g., malware families). This section discusses the existing malware detection research, which is roughly divided into three categories: 1) static-based malware detection approaches; 2) dynamic-based malware detection approaches; and 3) graph-based malware detection approaches.

2.1. Static-based malware detection approaches

Static analysis has been an active area of research in malware detection. Kang et al. (2016) attempted to automatically extract *n*-opcodes from the PE files and fed them into a machine learning model to detect malware. Takeuchi et al. (2018) adopted a Support Vector Machine (SVM) to tackle the frequency information of the opcode to identify attack behavior. Moreover, Hardy et al. (2016) trained Stacked AutoEncoders (SAEs) based on system calls, which showed advanced detection results compared with traditional machine learning methods. Afterward, Raff et al. (2018) utilized Convolutional Neural Networks (CNNs) to investigate bytecode grayscale graphs of malware,

although the accuracy is limited due to the fact that the raw bytecode often includes noise. Recently, to provide high throughput and low latency, Ahmed et al. (2022) developed a multi-layer CNN architecture based on malware grayscale images for IIoT malware classification, which leverages data preprocessing and transfer learning techniques to enhance the performance of traditional CNNs.

However, these static analysis approaches were feeble once the signature of new malware was not exposed in the predefined signature library. Moreover, such methods are vulnerable to code obfuscation and packing techniques (Li et al., 2020). In contrast, our proposed TS-Mal aims to detect new obfuscated malware variants by extracting fine-grained evolutionary information from the run-time API call events, which has the ability to capture the real malicious intention from the executive behavior rather than focusing on the obfuscated static signatures.

2.2. Dynamic-based malware detection approaches

Dynamic analysis is a common and practical scheme for malware detection (Kawakoya et al., 2019), which usually extracts API features to achieve detection purposes. In 2013, Elhadi et al. (2013) first represented each malware sample as an API call graph, then used the graph matching algorithm to calculate the similarity between the input sample and the malicious samples stored in the database, and finally identified the samples whose similarity was greater than the threshold as malware. To model the context relationships between APIs, Amer and Zelinka (2020) proposed a Markov chain-based malware detection model, which aims to represent the context semantics between malicious API call functions and benign API call functions as transformation matrixes for further detection. Considering that machine learning and deep learning can automatically extract malicious features from large-scale samples for detection, Uppal et al. (2014) propose a feature selection algorithm to select unique and different APIs and then apply multiple machine learning techniques (such as random forest, SVM, etc.) to classify malicious and benign PE files. Afterward, Pascanu et al. (2015), Kwon and Im (2017), and Li and Zheng (2021) all trained a recurrent neural network (RNN) based on API sequences to implement binary classification. However, their performance is limited by the length of the API sequence. Zhang et al. (2018) used principal component analysis to extract sensitive API calls and input features into multi-layer neural networks for malware detection. Then, to update existing machine learning-based malware detectors for better performance, Singh and Singh (2022) leveraged API calls to evaluate several basic parameters, such as k value, depth of tree, and so on, and finally obtained a high-precision malware classifier. In order to improve the detection accuracy, Han et al. (2021) used dynamic API sequence and ontology knowledge to detect APT malware. Maniriho et al. (2023) also proposed a new malware detection framework, API-MalDetect. API-MalDetect presents a hybrid feature extractor based on convolutional neural networks (CNN) and bidirectional gated loop units (BiGRUs) to extract high-level features from raw and long sequences of API calls, which has been experimentally proven to be effective in detecting invisible malware attacks. Recently, Cui et al. (2023) proposed API2Vec to solve the problem of interleaving API calls. They designed a temporal process graph (TPG) and a temporal API graph (TAG) to model inter-process and intra-process behavior, respectively, and then employed a heuristic random walk algorithm to generate malware behavior paths. Based on the path, a low-dimensional API embedding is eventually generated for further use in malware detection.

However, most of the traditional dynamic analysis approaches aim to extract isolated API features, which ignore the complex interactions among various malware objects, inevitably causing high false positives. Differently, our TS-Mal can precisely learn the comprehensive structural representations of the new malware, which benefits from modeling the various API category interactions by employing graph neural networks (GNNs).

2.3. Graph-based malware detection approaches

In recent years, graph neural networks (Fan et al., 2018; Wang and Philip, 2019; Ye et al., 2019) have been proposed for malware detection, which aim to model the interactive relationships between malware objects with heterogeneous graphs and formalize malware detection as a node classification task on graphs (Liu et al., 2023b). For instance, Fan et al. (2018) proposed Metagraph2vec, which learned the low-dimensional representations of the malware on the constructed heterogeneous information network (HIN) and identified the anomaly nodes. Wang and Philip (2019) designed MatchGNet, a graph-level-based malware detection framework that incorporates meta-path-based graph embedding to identify malware by measuring the graph-level similarities between benign samples and candidate samples. Additionally, Ye et al. (2019) presented an in-sample node embedding method and an out-sample node embedding method to contrastively learn the attack features of malware. Liu et al. (2021) made the first attempt to investigate the real-time detection framework, namely MG-DVD, which is a promising solution for real-time defense by employing two dynamic heterogeneous graph learning operators. To improve the robustness of malware detection, Ling et al. (2022) proposed MalGraph, which uses hierarchical graphs composed of function call graphs and control flow graphs to represent executable files and then trains an end-to-end GNN framework for malware detection. Experiments show that it is more robust against adversarial attacks. More recently, Chen et al. (2023) designed an expert system based on graph neural networks called MalwareExpert. They output detection results by indicating the most critical subgraphs that lead to malicious detection. In addition to having a promising detection accuracy, the retrieved binary semantic representations can be further used to explain the detection result and guide security analysts for better in-depth analysis.

However, the aforementioned GNN-based detection methods are not the most advanced solutions. On the one hand, they mostly followed a static graph that ignored the temporal dependence between the API sequences, inevitably incurring additional time costs to handle many invalid noise paths. Differently, our TS-Mal proposes a temporal vector learning method to effectively capture the evolutionary intents of attacks by analyzing temporal sequences of API calls. On the other hand, they strongly focused on the meta-path-based explicit neighborhood and simply compressed the sparse neighborhood to node representations, which is restricted to detection effectiveness. In this work, we design a dense-interactive graph embedding method to generate a fine-grained structural representation. This method has the ability to investigate comprehensive contextual semantic knowledge from the explicit and implicit neighbors to enhance the effectiveness of malware detection.

3. Preliminaries

In this section, we introduce some concepts used throughout this paper. First, we define the temporal API name call sequence and underlying API category interactions.

Definition 1. API name call sequence and API category interaction. An API name call sequence $\zeta = (API_1, API_2, \ldots, API_L)$ consists of the ordered API names called by the target software, which holds temporal evolutionary information. An API category interaction $e = (p, u, f)$ represents the edge between a target process (software) instance p and its neighbor u (e.g., a file, a system, a network, or a registry) associated with feature vector f, which holds contextual semantic information.

Based on the API category interactions in Definition 1, we introduce the malware heterogeneous graph in Definition 2.

Definition 2. Malware Events Heterogeneous Graph (MEHG) (Liu et al., 2022). The execution events of each malware can be formalized as a

Table 1
Notations used in TS-Mal framework.

Notation	Explanation
$\zeta = (API_1, API_2, \ldots, API_L)$	Ordered API name call sequence of the target software
$\mathcal{E} = (e_1, e_2, \ldots, e_L)$	Set of underlying API category interactions of the target software
$e_x = (p, u, f)$	API category interaction
$\boldsymbol{M} = \{M_1, M_2, \ldots, M_{\lvert M\rvert}\}$	Meta-paths set
$N^{(m)}$, $\hat{N}^{(m)}$	Original neighborhood and dense neighborhood guided by M_m, respectively
$\boldsymbol{h}_T$	Temporal vector of the target software
$\boldsymbol{h}_G$	Structural representation of the target software

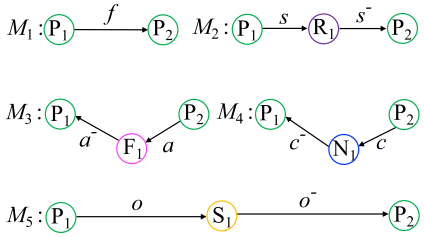


Fig. 2. Meta-paths in TS-Mal model.

heterogeneous graph $G = (V, E)$ with a node type mapping $\Psi : V \mapsto \mathcal{T}$ and an edge type mapping $\psi : E \mapsto \mathcal{R}$. Let V as the set of underlying API category nodes, E ⊆ V×V be the set of relationships between nodes in V. Each node $v \in V$ belongs to one particular underlying entity type in the node type set $\mathcal{T} : \Psi(v) \in \mathcal{T}$, and each edge $e \in E$ belongs to a particular relationship type in the edge type set $\mathcal{R} : \psi(e) \in \mathcal{R}$, where $|\mathcal{T}| + |\mathcal{R}| > 2$.

To explore the semantic information among malware objects and capture more discriminative attack patterns of various malware, we introduce meta-paths for dynamic malware variant detection, defined as below:

Definition 3. Meta-path (Sun et al., 2011). A meta-path is a template that specifies various types of relationships between two entities $v_1 \xrightarrow{R_1} v_2 \xrightarrow{R_2} \cdots \xrightarrow{R_i} v_n$, which defines a commuting relation $R = R_1 \diamond R_2 \diamond \cdots \diamond R_n$, where $\diamond$ represents the composition operation between node v_1 and v_n.

As shown in Fig. 2, we investigate 5 real-world meta-paths for malware detection. Different meta-paths represent different semantic information. For example, the meta-path M_3 depicts the semantic that "two processes access the same file", while the meta-path M_4 conveys the semantic that "two processes connect to the same network".

4. Methodology

In this section, we illustrate the details of the proposed TS-Mal model. As shown in Fig. 3, TS-Mal includes four modules: (1) Multi-dimensional Malware Behavior Resolving (i.e., module a); (2) TextRCNN-based Temporal Vector Learning (i.e., module b); (3) Dense-interactive GAT-based Structural Embedding (i.e., module c); and (4) Malware Family Detection (i.e., module d). For the sake of convenience, the key notations in this model are depicted in Table 1.

4.1. Multi-dimensional malware behavior resolving

The API call sequence generated by Cukoo Sandbox[1] consists of a series of API events exhibiting a coherent logical relationship. As shown in Table 2, each API operation represents a different function, and different attack types often correspond to different combinations of API calls.

[1] https://cuckoosandbox.org.

Table 2
The common Windows API operations in TS-Mal.

API operation	Functional Description
WriteProcessMemory	Write to memory in the specified process
ControlFile	Control file
NtQueryKey	Retrieve the stored buffer
AuxKlibInitialize	Initialize the secondary Kernel-Mode library
RegOpenKeyExA	Try requesting the KEY_READ value as the desired access mask
RegCloseKey	Release the handle to the specified key
GetAdaptersAddresses	Retrieve the address associated with the adapter on the local computer
NtClose	Close the specified handle

In order to capture more fine-grained attack features of malware, we first resolve the raw malware execution reports into temporal API name call sequences and interactions within underlying API categories, instead of conducting a rudimentary analysis of isolated temporal dependence. Given a raw malware execution report, the multi-dimensional malware behavior resolving module (module (a) in Fig. 3) can automatically divide the API names and API interaction behaviors into API call sequences and API category interaction sets, respectively.

The API name call sequences aim to capture the evolutionary patterns of malware attacks, and API category interaction is used to investigate the interconnections between malware entities. Additionally, for the API category interactions set $\mathcal{E}$, we extract five types of entities (i.e., process, file, network, system, and registry) and five types of relationships (i.e., process-process, process-file, process-network, process-system, and process-registry) among them. Afterward, we employ the heterogeneous graphs to powerfully model these malware entities and relationships, which can efficiently characterize the comprehensive context semantics of malware.

4.2. TextRCNN-based temporal vector learning

For a temporal API name sequence, it can be treated as a text, where each API name (e.g., WriteFile) is regarded as a word, so the malware temporal vector learning problem can be transformed into a text embedding problem. Considering the effectiveness of TextRCNN (Lai et al., 2015; Minaee et al., 2021; Wang and Li, 2022) in word embedding, this paper develops a TextRCNN-based temporal vector learning approach. Our motivation for using TextRCNN is twofold. On the one hand, there are strong or weak dependencies between successive API names. In this case, TextRCNN can leverage the forward and backward recursive structures to maximize the capture of such long-distance dependencies. On the other hand, each API name is a string of several words (such as WriteFile, ReadFile, etc.), and many API names contain the same words, so the convolutional layer in TextRCNN is efficient at capturing important string information. As illustrated in Fig. 3(b), this module involves the following two advanced components.

4.2.1. Duplicate subsequences filtering

In fact, software operation involves frequent API calls, including file operations, network behaviors, registry operations, etc. And most of the malware deliberately incorporates a substantial number of invalid API calls to increase the difficulty of detection. In this case, as shown in Fig. 5, the API sequence of some samples exhibits an exceptionally elon-

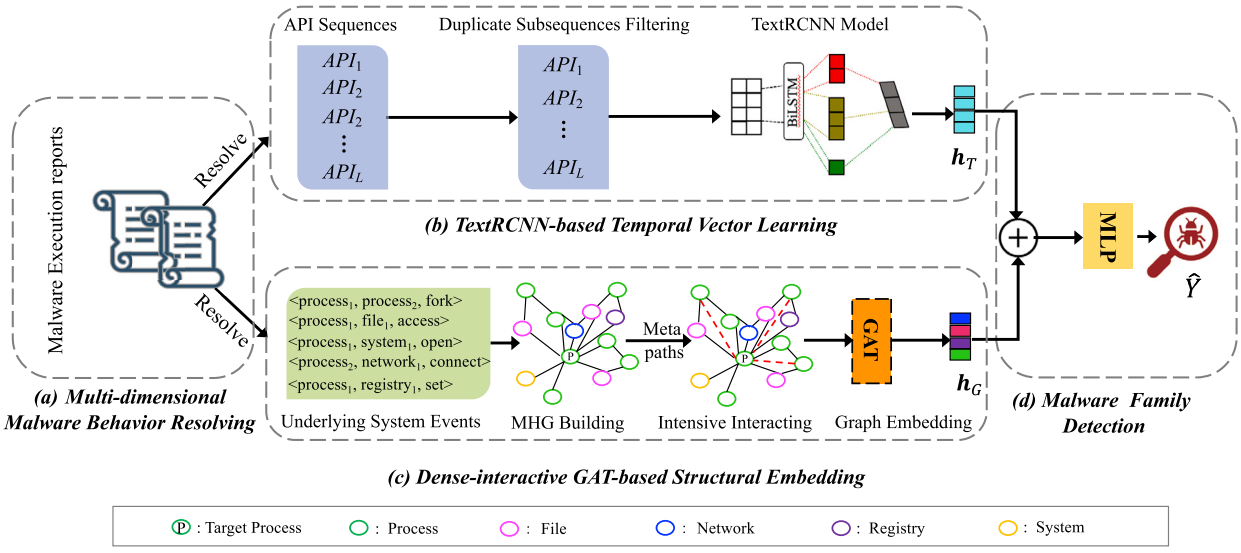


Fig. 3. TS-Mal workflow. (a) Multi-dimensional Malware Behavior Resolving aims to parse two dimensions of information, i.e., temporal API name sequences and underlying API category interactions from the execution API behavior of the target software. (b) TextRCNN-based Temporal Vector Learning accurately learns a temporal vector $\boldsymbol{h}_T$ by investigating the API name sequences. (c) Dense-interactive GAT-based Structural Embedding utilizes meta path-based implicit neighbors and graph attention networks to generate the structural representation $\boldsymbol{h}_G$ from the underlying API category interactions of the target software. (d) Malware Family Detection outputs the malware family $\hat{Y}$ by inputting the learned $\boldsymbol{h}_T$ and $\boldsymbol{h}_G$ of the target software. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

Algorithm 1: Duplicate subsequences filtering.

Input: Raw API name sequence $\zeta = (API_1, API_2, \ldots, API_L)$ of the target software
Output: Non-repetitive API name sequence ζ'

```
api_seq ← [];
for api_l in ζ do
    if api_seq == [] or api_l != api_seq[−1] then
        api_seq.append(api_l);
    else if api_l not in api_seq then
        api_seq.append(api_l);
ζ′ = "".join(api_seq).strip();
return ζ′
```

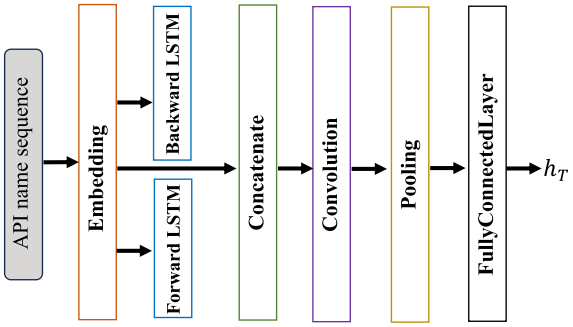


Fig. 4. The architecture of TextRCNN in TS-Mal.

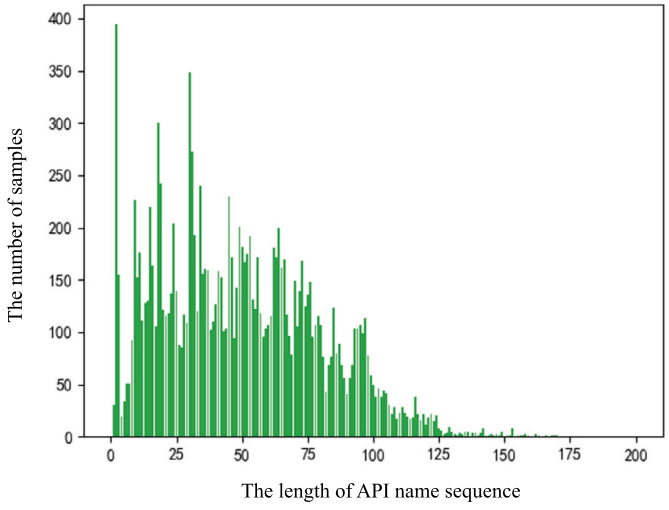


Fig. 5. The length distribution of API name sequence in TS-Mal.

gated pattern, which not only increases algorithm execution time but also compromises detection performance. Therefore, it is necessary to de-duplicate the original API name sequence. Sequential de-duplication can be implemented in two straightforward ways: by removing consecutive repeated calls and by enforcing strict de-duplication (retaining only the initial occurrence of each unique API) (Kim et al., 2016). Taking the call sequence "ABBBCCCNBA" as an example, the sequence "ABCNBA" is obtained by using the method of removing continuously repeated calls, and the sequence "ABCN" is obtained by using the method of strict de-duplication.

By utilizing Algorithm 1, we can obtain a non-repetitive API name sequence, which significantly reduces the training overhead of TS-Mal.

4.2.2. TextRCNN model

Here, we explain how to apply TextRCNN to capture the temporal vector of malware based on the tailored non-repetitive API name sequence ζ', where the input is a variable-length API name sequence and the output is a fixed-length temporal vector.

As shown in Fig. 4, the TextRCNN can be divided into six parts: embedding layer, Bi-LSTM layer, concatenate layer, convolution layer, maximum pooling layer, and fully connected layer. Concretely, the embedding layer converts the non-repetitive API name sequence into a word vector matrix; the Bi-LSTM layer utilizes a forward and backward recursive structure to get a forward and backward representation of each API, which fully captures long-term dependencies between successive APIs. Then, at each time step, the output of the Bi-LSTM layer is concatenated with the corresponding API embedding as the "semantic vector". To improve time efficiency, one-dimensional convolution ker-

nels with different widths (3, 4, 5) are used to carry out convolution operations and extract local features from the "semantic vector". The pooling layer conducts a max pooling operation on the convolution output to obtain three sequences, and then we input the above sequences to a fully connected layer to generate the final malware temporal vector h_T.

4.3. Dense-interactive structural embedding

To accurately identify the polymorphic malware, it is vital to associate the investigated temporal evolution patterns with the specific underlying contextual semantic information. To this end, we adopt dense-interactive neighbors and hierarchical graph attention networks (GATs) (Wang et al., 2019) to aggregate context information from the explicit and implicit interactions in malware heterogeneous graph, which is capable of capturing attack semantic knowledge from underlying API category interactions.

More specifically, as described in Fig. 3(c), given a constructed malware event heterogeneous graph (i.e., MEHG), the fine-grained structural representation can be obtained through the following steps:

1) Search the meta-path-based neighborhood $N^{(m)}$: To capture the semantic-unique malware attack patterns, TS-Mal leverages the predefined meta-paths in Fig. 2 to guide the random walk. Concretely, given the meta-path M_m, the neighborhood $N^{(m)}$ is:

$$N^{(m)} = \{u|(u,p) \in M_m, (p,u) \in M_m\}, \tag{1}$$

where $N^{(m)}$ represents all visited neighbor nodes u when the target process node p walks along with meta-path M_m.

2) Explore meta-graph-based dense neighborhood $\hat{N}^{(m)}$: On the basis of neighborhood $N^{(m)}$, we start to expend the 0-1 order neighbors as their dense implicit neighbors. Formally, the dense neighborhood $\hat{N}^{(m)}$ guided by meta-path M_m can be formulated as:

$$\hat{T}_p = [D^{(0)}(p), D_1^{(1)}(p), \cdots, D_i^{(1)}(p), \cdots], \tag{2}$$

$$\hat{N}^{(m)} = \bigcup \{\hat{T}_{nei_1}, \cdots, \hat{T}_{nei_j}, \cdots, \hat{T}_{nei_{|N^{(m)}|}}\}, \tag{3}$$

where $\hat{T}_p$ represents a dense neighborhood of each node p, $D^{(0)}(p)$ and $D_i^{(1)}(p)$ denote the 0-order neighbor and the i^{th} 1-order neighbor of node p.

3) Aggregate node-level representation $\mathbf{h}_p^{(m)(k)}$: We first obtain the node-level representation $\mathbf{h}_p^{(m)(k)}$ from the meta-path-based dense neighborhood $\hat{N}^{(m)}$. Formally, the node attention weight $\alpha_u^{(m)}$ and the k-th layer of the node-level aggregator are:

$$\alpha_u^{(m)} = \frac{exp(LeakyReLU(\mathbf{W}^T[\mathbf{X}_p, \mathbf{X}_u]))}{\sum_{u' \in \hat{N}^{(m)}} exp(LeakyReLU(\mathbf{W}^T[\mathbf{X}_p, \mathbf{X}_{u'}]))}, \tag{4}$$

$$\mathbf{h}_p^{(m)(k)} = \sigma\left((1+\epsilon^{(k)})\mathbf{h}_p^{(m)(k-1)} + \sum_{u \in \hat{N}^{(m)}} \alpha_u^{(m)} \mathbf{h}_u^{(m)(k-1)}\right), \tag{5}$$

where $k \in \{1, 2, \ldots, K\}$ denotes the index of the layer, $\boldsymbol{h}_p^{(m)(k-1)}$ and $\boldsymbol{h}_u^{(m)(k-1)}$ are the node-level representations of the target process node p and corresponding neighbor node u at the (k-1)-th layer, respectively. $\alpha_u^{(m)}$ represents the node attention weight of the neighbor node u in $\hat{N}^{(m)}$. $\epsilon_{(k)}$ is a balance parameter. Finally, the overall node-level representation of the target process node p concerning meta-path M_m is:

$$\mathbf{h}_p^{(m)} = CONCAT([\mathbf{h}_p^{(m)(k)}]_{k=1}^K). \tag{6}$$

4) Generate graph-level structural representation $\boldsymbol{h}_G$: Finally, we implement a summation *readout* function as the key operation to compute the graph-level representations over all nodes on the constructed heterogeneous graph G.

$$\mathbf{h}_G = READOUT(\mathbf{h}_v | v \in G). \tag{7}$$

4.4. Malware detection

In this subsection, inspired by the fact that the MLP classification layer is widely used in current malware detection research (Ye et al., 2019; Liu et al., 2023b), and because it has the advantage of adapting to various forms of input data and performing well in dealing with complex non-linear relationships, we train an MLP-based detector to demystify the multi-dimensional attack patterns of the target software and detect newly emerged malware. As illustrated in Fig. 3(d), we develop an MLP network to output the malware family $\hat{Y}$ (e.g., Trojan, AlienSpy) of the target software by inputting the learned temporal vector h_T and structural representation h_G, which is formulated as:

$$\hat{Y} = \sigma_d(w_d \cdot (\mathbf{h}_T \| \mathbf{h}_G) + b_d), \tag{8}$$

where $\|$ denotes the concatenation operation. σ_d is a nonlinear activation function, and w_d and b_d are the trainable weight parameters and bias of the detector, respectively. Finally, the loss function l of our TS-Mal can be formulated as:

$$l = ||(Y - \hat{Y}) \odot \mathcal{B}||_F^2, \tag{9}$$

where Y is the ground-truth label of the target software, and $\hat{Y}$ is the corresponding predicted label. With the loss function l, we perform stochastic gradient descent to fine-tune all learnable parameters.

5. Experiment

5.1. Dataset & experiment setup

5.1.1. Dataset

We validate the effectiveness of TS-Mal on three datasets, involving two public datasets (i.e., Kaggle Malware[2] and Mal-API-2019[3]) and our captured RANSOMWARE dataset.

- Kaggle Malware dataset (Schranko de Oliveira and Sassi, 2019; Setiawan et al., 2020) is the most widely used in Kaggle competitions, which includes the API sequences and labels of 37,784 malware and 1,079 benign software. Each API call sequence is extracted from the calls' elements in Cuckoo Sandbox reports. It consists of 307 distinct API call values coded between 0 and 306.
- Mal-API-2019 dataset (Catak and Yazı, 2019; Catak et al., 2021) involves a total of 7,107 malware, and each record of this dataset is an ordered API sequence generated by the Cuckoo sandbox environment.
- The RANSOMWARE dataset collected 13,887 samples spanning from Jun 2021 to Dec 2022 from the authoritative VirusShare,[4] each of which contains a Cuckoo sandbox behavior report and a label generated by VirusTotal.[5]

The basic statistics for all datasets are presented in Table 3. For all datasets, we randomly select 60% of the samples as the training set, 20% of the samples as the verification set, and the rest of the samples as the test set.

5.1.2. Experiment setup

We default to a learning rate of 0.01 and drop out of 0.5, and the specific parameters of the five parts of TextRCNN are shown in Table 4. Furthermore, we train TS-Mal on a machine with an Intel(R) Xeon(R) Silver 4214R CPU @3.40 GHz with 64 GB RAM and 4 × NVIDIA Tesla

[2] https://www.kaggle.com/ang3loliveira/malware-analysis-datasets-api-call-sequences.
[3] https://github.com/ocatak/malware_api_class.
[4] https://virusshare.com.
[5] https://www.virustotal.com.

Table 3
Statistics of the three datasets.

Dataset	Samples Distribution								
Kaggle Malware	Trojan	Downloader	Virus	Spyware	Adware	Dropper	Worm	Backdoor	Benign
	12,824	6,560	5,522	5,897	4,449	945	808	779	1,079
Mal-API-2019	Trojan	Downloader	Worms	Virus	Backdoor	Dropper	Spyware	Adware	-
	1,001	1,001	1,001	1,001	1,001	891	832	379	-
RANSOMWARE	Globelmposter	Stop	Avaddon	Ryuk	WannaCry	Sodinokibi	Phobos	Benign	-
	502	1,196	820	100	4,289	515	1,487	4,978	-

Table 4
The major parameters of TextRCNN in TS-Mal.

Layer	Parameter	Value
Embedding layer	*Embedding_dim*	100
Bi-LSTM layer	*hidden_dim*	128
Convolutional layer	*Number_layer*	3
	Number_input_pipes	200
	Number_output_pipes	100
Polling layer	*Polling_way*	AdaptiveAvgPool1d
Fully connected layer	*Activation*	relu

Table 5
The illustration of evaluation metrics.

Metric	Description
TP	The number of malware variants that are classified correctly as malware
TN	The number of benign software that are classified correctly as benign
FN	The number of malware variants that are misjudged as benign
FP	The number of benign software that are misjudged as malware
ACC	$(TP+TN)/(TP+TN+FP+FN)$
Precision	$TP/(TP+FP)$
Recall	$TP/(TP+FN)$
F1-score	$2\times Precision\times Recall/(Precision+Recall)$
AUC	Area under *ROC* curve

K80 GPU. All of the experiments developed with Python 3.6 are executed on the pytorch (1.10.0+cu113) framework supported by the CentOS Linux release 7.6.1810 (Core) operating system.

We measure the performance of our proposed framework on malware detection tasks in terms of accuracy (ACC), precision, recall, F1-score, and AUC, which are widely used in previous works, and corresponding descriptions are shown in Table 5.

5.2. Effectiveness and efficiency analysis

In this subsection, we compare the effectiveness and efficiency of our proposed TS-Mal and the six baseline methods for malware detection tasks. Concretely, SVM (Takeuchi et al., 2018) and MalConv (Raff et al., 2018) are both typical static detection methods, especially MalConv, which is the most effective static-based malware detection method so far. LSTM (Setiawan et al., 2020) and CNN+BPNN (Zhang et al., 2019) are both the latest dynamic API-based malware detection approaches. Hawk (Hei et al., 2021) and MatchGNet (Wang and Philip, 2019) belong to graph-based malware detection approaches, which are both state-of-the-art research. Concretely, we run Mal-TS 10 times and report the average results.

5.2.1. Effectiveness evaluation

The comparison results are shown in Table 6 and Fig. 6. According to the results, we have the following observations:

Firstly, the proposed TS-Mal outperforms the six baseline methods on all datasets. Particularly, the proposed TS-Mal achieves improvements of at least 2.4%, 6.1%, and 6.4% in accuracy (i.e., ACC) on the RANSOMWARE dataset, Kaggle Malware dataset, and Mal-API-2019 dataset, respectively. The experimental results demonstrate that our proposed temporal learning module has an excellent ability to simultaneously capture temporal attack patterns along with its underlying

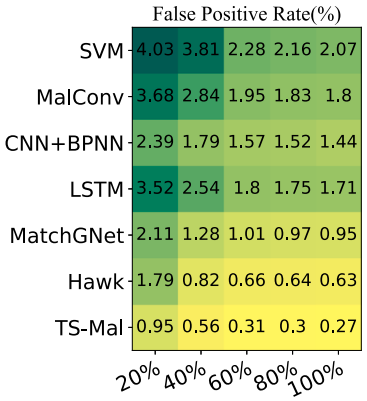


Fig. 6. False positive rate of seven methods with different proportion of training data.

semantic features, which significantly enhances the accuracy of detecting malware attacks.

Secondly, graph-based methods (i.e., MatchGNet, Hawk, and TS-Mal) exhibit superior effectiveness in malware detection compared to traditional machine learning or deep learning approaches. This is attributed to the utilization of graph neural networks (GNNs) for capturing high-order semantic information from diverse API category interactions, which can better express the comprehensive representations of the new malware variants.

Thirdly, as shown in Fig. 6, our proposed TS-Mal shows the lowest false positive rate, whereas traditional machine learning and deep learning methods demonstrate significantly higher false positive rates compared to TS-Mal. The reason behind this is that TS-Mal comprehensively incorporates the multi-dimensional features of malware, including temporal features and structural features, which can effectively distinguish malicious software from benign software. However, traditional machine learning and deep learning techniques such as SVM, MalConv, LSTM, and CNN+BPNN solely extract isolated static features or temporal features without fully characterizing polymorphic malware, which inevitably brings high false positives.

5.2.2. Efficiency evaluation

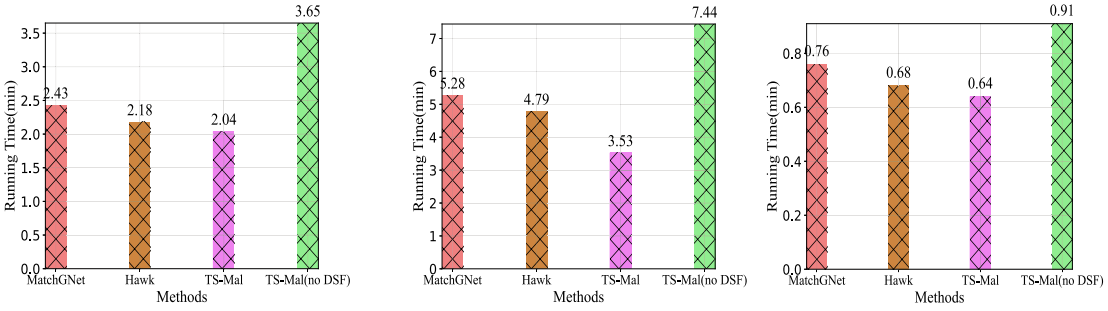
In this subsection, we mainly compare the time efficiency of TS-Mal with the graph-based detection methods (e.g., MatchGNet and Hawk) and its variant that drops the duplicate subsequences filtering (DSF) module. The comparison results on three datasets are shown in Fig. 7, and the following observation can be highlighted:

Firstly, as shown in Figs. 7(a-c), the runtimes of TS-Mal are significantly less than those of graph-based detection methods (i.e., MatchGNet and Hawk) across all datasets. This can be attributed to the fact that our proposed TS-Mal differentiates API name call sequence and underlying API category interactions rather than amalgamating all entities into a large-scale heterogeneous graph, which saves walking and training overhead.

Secondly, we find that the proposed TS-Mal is 1.79×, 2.11×, and 1.42× faster than its variant model TS-Mal (without DSF) on the RANSOMWARE dataset in Fig. 7(a), Kaggle Malware dataset in Fig. 7(b), and Mal-API-2019 dataset in Fig. 7(c), respectively. This shows that the

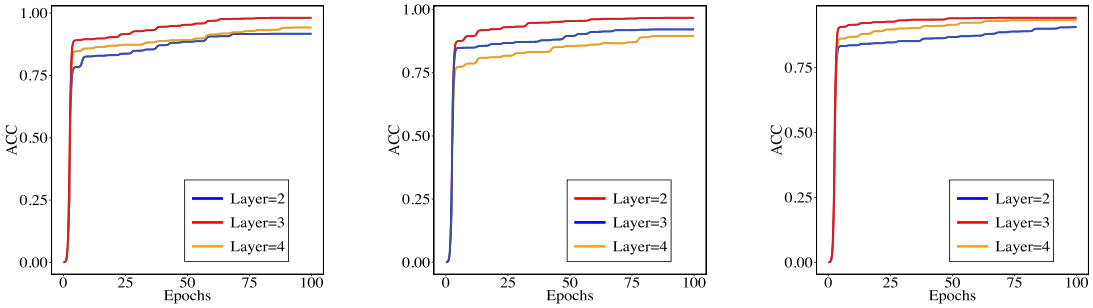
Table 6
Performance on malware detection.

Method	RANSOMWARE dataset					Kaggle Malware dataset					Mal-API-2019 dataset				
	Recall	Precision	ACC	F1-score	AUC	Recall	Precision	ACC	F1-score	AUC	Recall	Precision	ACC	F1-score	AUC
SVM (Takeuchi et al., 2018)	0.806	0.805	0.806	0.806	0.797	0.763	0.772	0.769	0.767	0.725	0.724	0.741	0.739	0.732	0.682
MalConv (Raff et al., 2018)	0.829	0.831	0.829	0.830	0.792	0.805	0.819	0.813	0.812	0.769	0.766	0.774	0.771	0.770	0.728
CNN+BPNN (Zhang et al., 2019)	0.891	0.902	0.897	0.896	0.864	0.879	0.892	0.887	0.885	0.840	0.834	0.848	0.845	0.841	0.809
LSTM (Setiawan et al., 2020)	0.853	0.869	0.865	0.861	0.808	0.834	0.842	0.839	0.838	0.801	0.802	0.816	0.812	0.809	0.777
MatchGNet (Wang and Philip, 2019)	0.924	0.933	0.929	0.928	0.887	0.891	0.896	0.894	0.893	0.853	0.862	0.874	0.873	0.868	0.825
Hawk (Hei et al., 2021)	0.932	0.939	0.932	0.935	0.901	0.906	0.915	0.906	0.910	0.883	0.878	0.879	0.878	0.878	0.845
TS-Mal	**0.955**	**0.962**	**0.956**	**0.958**	**0.926**	**0.958**	**0.971**	**0.967**	**0.964**	**0.934**	**0.930**	**0.944**	**0.942**	**0.937**	**0.901**
Improvement	2.3%	2.3%	2.4%	2.3%	2.5%	5.2%	5.6%	6.1%	5.4%	5.1%	5.2%	6.5%	6.4%	5.9%	5.6%



(a) RANSOMWARE dataset (b) Kaggle Malware dataset (c) Mal-API-2019 dataset

Fig. 7. Running time of baseline methods, TS-Mal, and its variant TS-Mal (no DSF) on three datasets, respectively.



(a) RANSOMWARE dataset (b) Kaggle Malware dataset (c) Mal-API-2019 dataset

Fig. 8. Performance change of TS-Mal with different number of layers in terms of ACC.

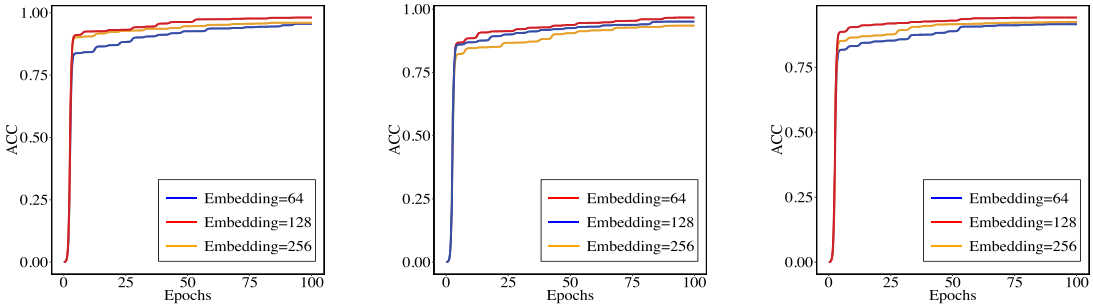
duplicate subsequences filtering module is crucial and effective for filtering redundant attack information and improving real-time detection efficiency.

5.3. Parameter sensitivity analysis

In this subsection, we conduct ample experiments to investigate the sensitivity of several essential parameters in TS-Mal. We mainly focus on these parameters, including the number of GAT layers (i.e., K) and the embedding size of structural representation (i.e., d).

Effect of The Number of Layers *K*. We first investigate the effect of the layer number (i.e., K) of the graph attention networks (GATs) in TS-Mal. As shown in Figs. 8(a-c), ACC increases with the increase of K, demonstrating that the larger the number of layers; the more high-order neighborhood information can be carried. However, as K exceeds an appropriate value, the detection accuracy declines due to GAT layers aggregating more noise nodes to represent target nodes, degrading the performance of graph embedding and malware detection.

Effect of Embedding Size *d*. We further analyze the effect of the embedding size d of the low-dimensional structural representation in TS-Mal on different datasets. We restrict other parameters and vary d from 64 to 256. As depicted in Figs. 9(a-c), the detection accuracies of TS-Mal are highest at embedding size d set to 128, 64, and 128 on



(a) RANSOMWARE dataset (b) Kaggle Malware dataset (c) Mal-API-2019 dataset

Fig. 9. Performance change of TS-Mal with different embedding size in terms of ACC.

Table 7
The ACC performance comparison between TS-Mal and its variants.

Method	RANSOMWARE ACC	Kaggle Malware ACC	Mal-API-2019 ACC
TS-Mal$_{_TV}$	0.922	0.936	0.905
TS-Mal$_{_SR}$	0.917	0.909	0.896
TS-Mal$_{_IIN}$	0.938	0.943	0.930
TS-Mal$_{_BiLSTM}$	0.931	0.961	0.938
TS-Mal$_{_Transformer}$	0.925	0.954	0.931
TS-Mal	0.956	0.967	0.942

the RANSOMWARE dataset, Kaggle Malware dataset, and Mal-API-2019 dataset, respectively.

5.4. Ablation study

In this subsection, we perform comprehensive experiments to investigate the effects of different components on TS-Mal through ablation studies.

Temporal Vector. TextRCNN-based temporal vector learning is a crucial component in TS-Mal. We first study its contribution by comparing TS-Mal with the variant model TS-Mal$_{TV}$ without temporal vector (TV). As illustrated in Table 7, compared to TS-Mal$_{_TV}$, the proposed TS-Mal improves 3.4% on the RANSOMWARE dataset, 3.1% on the Kaggle Malware dataset, and 3.7% on the Mal-API-2019 dataset, respectively. The experimental results demonstrate that our proposed temporal vector learning module effectively captures the discernible evolutionary attack patterns of malware from the API name sequences, thereby facilitating the detection of newly emerged malware.

Structural Representation. Subsequently, we explore the role of dense-interactive GAT-based structural embedding in TS-Mal. From Table 7, we observe that the ACC performance of TS-Mal is superior to that of its variant model, TS-Mal$_{_SR}$ without structural representation (SR). This can be attributed to the comprehensive semantics between the underlying API categories, which can capture more comprehensive interdependence patterns of malware, enhancing the detection performance for stealthy malware.

Intensive Interacting Neighborhoods. We further investigate the impact of intensive interacting neighborhoods on TS-Mal by comparing it with its variant TS-Mal$_{_IIN}$ without intensive interacting neighborhoods (IIN). As evidenced by the results presented in Table 7, the detection performance of TS-Mal surpasses that of TS-Mal$_{_IIN}$, which evinces that the implicit interactions among crucial node pairs can provide valuable insights to facilitate the identification of novel malware variants.

TextRCNN. We finally assess the contribution of the TextRCNN in TS-Mal by comparing it with its variants, TS-Mal$_{_BiLSTM}$ and TS-Mal$_{_Transformer}$. We found the TS-Mal to be slightly superior to both variant models in Table 7, especially the TS-Mal with Transformer. These phenomena may be attributed to the fact that, compared with the traditional Bi-LSTM, which can only capture the temporal dependencies between API name sequences, TS-Mal has the ability to utilize the TextRCNN to capture the forward and backward temporal dependencies of API name sequences and characterize the *n*-gram information of API strings. This helps improve the expression of malware temporal vectors. What's more, unlike Transformer, which relies entirely on the attention mechanism to capture global dependencies between API name sequences, TS-Mal utilizes TextRCNN's bidirectional recursive architecture to capture such contextual dependencies, which is more in line with the long-distance dependencies of API sequences.

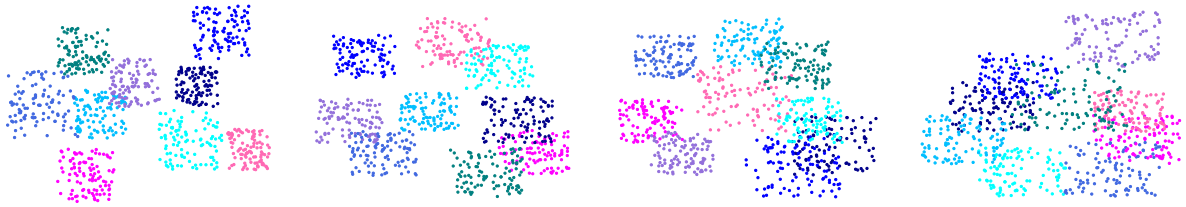
5.5. Visualization analysis

To more clearly and directly verify the effectiveness of proposed fusion features (i.e., temporal representation and structural representation), in this subsection, we employ t-Distributed Stochastic Neighbor Embedding (t-SNE (Van der Maaten and Hinton, 2008)) to visually represent the distribution of the low-dimensional embedding vectors generated by TS-Mal and its variant models on the RANSOMWARE dataset. This can also help better understand the distribution of various malware families and improve the interpretability of detection results. Empirically, the same type of malware should be mapped into closer proximity in the learned embedding space.

Fig. 10 shows the visualization of Hawk, TS-Mal, and two variant models of TS-Mal, respectively. As shown in Fig. 10(c), due to the inability of the T-Mal model to incorporate GAT for capturing contextual semantic patterns within the underlying API categories, the malware samples belonging to different types are intricately embedded in the low-dimensional space, and the boundaries of different types of malware are blurred. Consequently, it is incapable of accurately detecting new malware as the embedding features derived from genuine attack behaviors are intertwined with those of normal behaviors. As illustrated in Fig. 10(a), we further find that the TS-Mal model exhibits easily distinguishable malware representations, whereas the S-Mal shown in Fig. 10(b) shows suboptimal performance. This observation highlights the superior performance of integrated embeddings in TS-Mal compared to isolated structural representations. As demonstrated in Fig. 10(d), Hawk exhibits the worst performance due to its inability to differentiate the API behavior of malware. Consequently, it fails to model the temporal features of malware and capture the intricate interacting relationships between malware objects.

6. Conclusion

Security companies increasingly rely on efficient defensive techniques to enhance resilience against cyber attacks. To settle the problem that existing API-based research merely investigates the temporal dependencies of successive APIs but ignores the API string *n*-gram information and the contextual semantics derived from the underlying API categories, which leads to insufficient detection performance. In this paper, TS-Mal, a novel malware detection model, is proposed to automatically extract malware features from temporal API name sequences and underlying API category interactions. More specifically, temporal



(a) TS-Mal (b) S-Mal (c) T-Mal (d) Hawk

Fig. 10. t-SEN visualizations of TS-Mal and baseline methods on the RANSOMWARE dataset. (a) and (d) are visualizations for TS-Mal and Hawk, respectively; (b) and (c) show the S-Mal with the single structural representation and the T-Mal with the single temporal vector, respectively.

vector learning is presented to capture malware's temporal attack vectors. Then, to further depict the fine-grained attack structure semantics of attack events, a dense-interactive structural embedding is proposed, which is capable of incorporating dense explicit and implicit interactions between crucial node pairs to boost the detection effectiveness of polymorphic malware. Experiment results prove that our proposed TS-Mal outperforms state-of-the-art methods in malware detection tasks, which helps security experts reveal the attack intention of the polymorphic malware from different perspectives.

However, our proposed TS-Mal relies on sufficient training samples and labeling information; hence, the future direction of research will explore an online few-shot malware detection framework, which will provide a proactive and real-time cyber security defense.

CRediT authorship contribution statement

Wanyu Li: Conceptualization, Data curation, Investigation, Methodology, Software, Validation, Writing – original draft. **Hailiang Tang:** Resources, Supervision, Writing – review & editing. **Hailin Zhu:** Funding acquisition, Project administration, Supervision. **Wenxiao Zhang:** Software, Validation, Visualization. **Chen Liu:** Validation, Writing – review & editing.

Declaration of competing interest

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

Data availability

Data will be made available on request.

Acknowledgements

The authors would like to thank the anonymous reviewers for their careful work and constructive comments on this paper.

References

Abdelnabi, S., Krombholz, K., Fritz, M., 2020. VisualPhishNet: zero-day phishing website detection by visual similarity. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, pp. 1681–1698.

Ahmed, I., Anisetti, M., Ahmad, A., Jeon, G., 2022. A multilayer deep learning approach for malware classification in 5G-enabled IIoT. IEEE Trans. Ind. Inform. 19, 1495–1503.

Amer, E., Zelinka, I., 2020. A dynamic windows malware detection and prediction method based on contextual understanding of api call sequence. Comput. Secur. 92, 101760.

Catak, F.O., Ahmed, J., Sahinbas, K., Khand, Z.H., 2021. Data augmentation based malware detection using convolutional neural networks. PeerJ Comput. Sci. 7, e346. https://doi.org/10.7717/peerj-cs.346.

Catak, F.O., Yazı, A.F., 2019. A benchmark API call dataset for windows PE malware classification. arXiv preprint arXiv:1905.01999.

Chen, X., Li, C., Wang, D., Wen, S., Zhang, J., Nepal, S., Xiang, Y., Ren, K., 2019. Android HIV: a study of repackaging malware for evading machine-learning detection. IEEE Trans. Inf. Forensics Secur. 15, 987–1001.

Chen, Y.H., Lin, S.C., Huang, S.C., Lei, C.L., Huang, C.Y., 2023. Guided malware sample analysis based on graph neural networks. IEEE Trans. Inf. Forensics Secur.

Cui, L., Cui, J., Ji, Y., Hao, Z., Li, L., Ding, Z., 2023. API2Vec: learning representations of api sequences for malware detection. In: Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 261–273.

Elhadi, A.A.E., Maarof, M.A., Barry, B., 2013. Improving the detection of malware behaviour using simplified data dependent api call graph. Int. J. Netw. Secur. Appl. 7, 29–42.

Fan, Y., Hou, S., Zhang, Y., Ye, Y., Abdulhayoglu, M., 2018. Gotcha-sly malware! Scorpion a metagraph2vec based malware detection system. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 253–262.

Han, W., Xue, J., Wang, Y., Zhang, F., Gao, X., 2021. APTMalinsight: identify and cognize APT malware based on system call information and ontology knowledge framework. Inf. Sci. 546, 633–664.

Hardy, W., Chen, L., Hou, S., Ye, Y., Li, X., 2016. DL4MD: a deep learning framework for intelligent malware detection. In: Proceedings of the International Conference on Data Science (ICDATA), the Steering Committee of the World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), p. 61.

Hei, Y., Yang, R., Peng, H., Wang, L., Xu, X., Liu, J., Liu, H., Xu, J., Sun, L., 2021. Hawk: rapid Android malware detection through heterogeneous graph attention networks. IEEE Trans. Neural Netw. Learn. Syst.

Kang, B., Yerima, S.Y., McLaughlin, K., Sezer, S., 2016. N-opcode analysis for Android malware classification and categorization. In: 2016 International Conference on Cyber Security and Protection of Digital Services (Cyber Security). IEEE, pp. 1–7.

Kawakoya, Y., Shioji, E., Iwamura, M., Miyoshi, J., 2019. API Chaser: taint-assisted sandbox for evasive malware analysis. J. Inf. Process. 27, 297–314.

Kim, H.J., Kim, J.H., Kim, J.T., Kim, I.K., Chung, T.M., 2016. Feature-chain based malware detection using multiple sequence alignment of api call. IEICE Trans. Inf. Syst. 99, 1071–1080.

Kwon, I., Im, E.G., 2017. Extracting the representative api call patterns of malware families using recurrent neural network. In: Proceedings of the International Conference on Research in Adaptive and Convergent Systems, pp. 202–207.

Lai, S., Xu, L., Liu, K., Zhao, J., 2015. Recurrent convolutional neural networks for text classification. In: Proceedings of the AAAI Conference on Artificial Intelligence.

Li, C., Peng, H., Li, J., Sun, L., Lyu, L., Wang, L., Yu, P.S., He, L., 2022. Joint stance and rumor detection in hierarchical heterogeneous graph. IEEE Trans. Neural Netw. Learn. Syst. 33, 2530–2542. https://doi.org/10.1109/TNNLS.2021.3114027.

Li, C., Zheng, J., 2021. Api call-based malware classification using recurrent neural networks. J. Cyber Secur. Mobil., 617–640.

Li, Y., Wang, Y., Wang, Y., Ke, L., Tan, Y.a., 2020. A feature-vector generative adversarial network for evading PDF malware classifiers. Inf. Sci. 523, 38–48.

Ling, X., Wu, L., Deng, W., Qu, Z., Zhang, J., Zhang, S., Ma, T., Wang, B., Wu, C., Ji, S., 2022. MalGraph: hierarchical graph neural networks for robust windows malware detection. In: IEEE INFOCOM 2022-IEEE Conference on Computer Communications. IEEE, pp. 1998–2007.

Liu, C., Li, B., Zhao, J., Liu, X., Li, C., 2023a. MalAF: malware attack foretelling from run-time behavior graph sequence. IEEE Trans. Dependable Secure Comput.

Liu, C., Li, B., Zhao, J., Su, M., Liu, X.D., 2021. MG-DVD: a real-time framework for malware variant detection based on dynamic heterogeneous graph learning. In: Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence. IJCAI-21, pp. 1512–1519.

Liu, C., Li, B., Zhao, J., Zhen, Z., Feng, W., Liu, X., 2023b. TI-MVD: a temporal interaction-enhanced model for malware variants detection. Knowl.-Based Syst. 110850.

Liu, C., Li, B., Zhao, J., Zhen, Z., Liu, X., Zhang, Q., 2022. FewM-HGCL: few-shot malware variants detection via heterogeneous graph contrastive learning. IEEE Trans. Dependable Secure Comput.

Van der Maaten, L., Hinton, G., 2008. Visualizing data using t-SNE. J. Mach. Learn. Res. 9.

Maniriho, P., Mahmood, A.N., Chowdhury, M.J.M., 2023. API-MalDetect: automated malware detection framework for windows based on API calls and deep learning techniques. J. Netw. Comput. Appl. 218, 103704.

Minaee, S., Kalchbrenner, N., Cambria, E., Nikzad, N., Chenaghlu, M., Gao, J., 2021. Deep learning–based text classification: a comprehensive review. ACM Comput. Surv. 54, 1–40.

Schranko de Oliveira, A., Sassi, R.J., 2019. Behavioral malware detection using deep graph convolutional neural networks.

Pascanu, R., Stokes, J.W., Sanossian, H., Marinescu, M., Thomas, A., 2015. Malware classification with recurrent networks. In: 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, pp. 1916–1920.
Gaviria de la Puerta, J., Sanz, B., 2017. Using Dalvik opcodes for malware detection on Android. Log. J. IGPL 25, 938–948.
Qu, Y., Bai, T., Zhang, W., Nie, J., Tang, J., 2019. An end-to-end neighborhood-based interaction model for knowledge-enhanced recommendation. In: Proceedings of the 1st International Workshop on Deep Learning Practice for High-Dimensional Sparse Data, pp. 1–9.
Raff, E., Barker, J., Sylvester, J., Brandon, R., Catanzaro, B., Nicholas, C.K., 2018. Malware detection by eating a whole EXE. In: Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence.
Setiawan, H., Putro, P.A.W., Pramadi, Y.R., et al., 2020. Comparison of LSTM architecture for malware classification. In: 2020 International Conference on Informatics, Multimedia, Cyber and Information System (ICIMCIS). IEEE, pp. 93–97.
Singh, J., Singh, J., 2022. Assessment of supervised machine learning algorithms using dynamic api calls for malware detection. Int. J. Comput. Appl. 44, 270–277.
Sophos, 2022. The state of ransomware 2022. https://www.sophos.com/en-us/content/state-of-ransomware.
Sun, M., Li, X., Lui, J.C., Ma, R.T., Liang, Z., 2016. Monet: a user-oriented behavior-based malware variants detection system for Android. IEEE Trans. Inf. Forensics Secur. 12, 1103–1112.
Sun, Y., Han, J., Yan, X., Yu, P.S., Wu, T., 2011. PathSim: meta path-based top-k similarity search in heterogeneous information networks. Proc. VLDB Endow. 4, 992–1003.
Takeuchi, Y., Sakai, K., Fukumoto, S., 2018. Detecting ransomware using support vector machines. In: Workshop Proceedings of the 47th International Conference on Parallel Processing, pp. 1–6.
Uppal, D., Sinha, R., Mehra, V., Jain, V., 2014. Exploring behavioral aspects of api calls for malware identification and categorization. In: 2014 International Conference on Computational Intelligence and Communication Networks. IEEE, pp. 824–828.
Wang, Q., Hassan, W.U., Li, D., Jee, K., Yu, X., Zou, K., Rhee, J., Chen, Z., Cheng, W., Gunter, C.A., et al., 2020. You are what you do: hunting stealthy malware via data provenance analysis. In: NDSS.
Wang, Q., Li, X., 2022. Chinese news title classification model based on ERNIE-TextRCNN. In: Proceedings of the 2022 5th International Conference on Machine Learning and Natural Language Processing, pp. 147–151.
Wang, S., Philip, S.Y., 2019. Heterogeneous graph matching networks: application to unknown malware detection. In: 2019 IEEE International Conference on Big Data (Big Data). IEEE, pp. 5401–5408.
Wang, X., Ji, H., Shi, C., Wang, B., Ye, Y., Cui, P., Yu, P.S., 2019. Heterogeneous graph attention network. In: The World Wide Web Conference, pp. 2022–2032.
Ye, Y., Hou, S., Chen, L., Lei, J., Wan, W., Wang, J., Xiong, Q., Shao, F., 2019. Out-of-sample node representation learning for heterogeneous graph in real-time Android malware detection. In: 28th International Joint Conference on Artificial Intelligence (IJCAI).
Zhang, J., Qin, Z., Yin, H., Ou, L., Zhang, K., 2019. A feature-hybrid malware variants detection using CNN based opcode embedding and bpnn based API embedding. Comput. Secur. 84, 376–392.
Zhang, J., Zhang, K., Qin, Z., Yin, H., Wu, Q., 2018. Sensitive system calls based packed malware variants detection using principal component initialized multilayers neural networks. Cybersecurity 1, 1–13.
Zhang, Z., Qi, P., Wang, W., 2020. Dynamic malware analysis with feature engineering and feature learning. In: Proceedings of the AAAI Conference on Artificial Intelligence, pp. 1210–1217.

Wanyu Li received the M.A. degree in Administrative Management and E-government from Shandong Normal University, China, in 2014. She is currently an assistant professor at Qilu Normal University, China. Her current main research interests include Artificial Intelligence, Deep Learning and Metaverse.

Hailiang Tang is an Assistant Professor in School of Information Science and Engineering, Qilu Normal University, China. He received the PhD degree from Shandong Normal University in 2021. His current research interests include big data, NLP, and industrial information security.

Hailin Zhu received the M.A. degree in Computer Software and Theory from Shandong Normal University, China, in 2009. He is currently an assistant professor with the College of Information Science and Engineering, Qilu Normal University, China. His current main research interests include Machine Learning, Pattern Recognition and Internet of Things.

Wenxiao Zhang is a PHD candidate in the Department of Computer Science and Engineering, Shandong Normal University, China. She received the BS and MS degree in Information Science and Engineering from Shandong Normal University in 2017. Her research interests include anomaly detection, graph mining, event detection and forecasting, optimization algorithm.

Chen Liu is a PHD candidate in the School of Computer Science and Engineering, Beihang University, China. Her research interests include malware detection, graph neural networks, and anomaly detection.