Hindawi
Security and Communication Networks
Volume 2021, Article ID 8077220, 14 pages
https://doi.org/10.1155/2021/8077220

Research Article

Toward Identifying APT Malware through API System Calls

Chaoxian Wei,[1] **Qiang Li**,[2] **Dong Guo**,[3] **and Xiangyu Meng**[3]

[1]*College of Software, Jilin University, Changchun 130012, China*
[2]*College of Computer Science and Technology, Jilin University, Changchun 130012, China*
[3]*Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Jilin University, Changchun 130012, China*

Correspondence should be addressed to Xiangyu Meng; xiangyumeng@jlu.edu.cn

Received 2 August 2021; Revised 28 October 2021; Accepted 23 November 2021; Published 9 December 2021

Academic Editor: Angel M. Del Rey

Self-developed malware was usually used by advanced persistent threat (APT) attackers to launch APT attacks. Therefore, we can enhance the understanding and cognition of APT attacks by comprehending the behavior of APT malware. Unfortunately, the current research cannot effectively explain the relationship between the recognition, detection, and defense of APT. The model of similar studies also lacks an explanation about it. To defend against APT attacks and inquire about the similarity of different APT attacks, this study proposes an APT malware classification method based on a combination of multiple deep learning algorithms and transfer learning by collecting malware used in several famous APT groups in public. By extracting the application programming interface (API) system calls, with the vector representation of features by combining dynamic LSTM and attention algorithm, we can obtain API at different APT families classification contributions trained dynamic. Thus, we used transfer learning to perform multiple classifications of the APT family. This study aims to reduce the burden of network security staff from reviewing a large number of suspicious files when defending against APT attacks. Additionally, it can effectively intercept them in the initial invasion stage of APT to perform targeted defense against specific APT attacks by combining threat intelligence in public. The experimental result shows that the proposed method can achieve 99.2% in distinguishing common malware from APT malware and assign APT malware to different APT families with an accuracy of 95.5%.

1. Introduction

Recently, advanced persistent threat (APT) attacks have been continuously developed, and new types of APT emerge, posing severe threats and challenges to the network security environment in the present world. They usually obtain huge financial or technical input, and they often perform long-term and complex attacks on certain targets [1]. The purpose is to steal valuable confidential data or perform network espionage activities, which will cause severe harm; thus, research on APT detection and prevention is urgent.

Similar to traditional network attacks, APT attackers must use malware as attack weapons to attack in cyberspace [2]. However, unlike traditional network attacks, APT attacks will use some independent development malware to achieve specific purposes against different targets [3]. This malware is collectively called APT malware [4]. APT malware is one kind of advanced malware tailored for special targets, which has posed even more serious threats than the traditional malware [2]. Compared with other malware, the APT-type attacks follow a different attack roadmap [5]; and APT malware is very different from ordinary malware. The primary purpose of APT malware is to remotely control the machines and to steal confidential data, rather than to launch denial-of-service attacks, send spam emails, or cause damage. It requires a high degree of stealth over a prolonged duration of operation. For example, in the case of those bots and worms, the attackers need to use the command and control servers to remotely control thousands of infected hosts. But APT attackers do not use the same C&C server to remotely control so many infected end-user machines, because it would increase the risk of exposure. The crafted malware is only used for the end-user machines which are valuable to them [6]. APT attackers will evolve their weapons, making the existing automated security measures in the face of sudden APT attack unable to accurately identify

the suspicious file detected by the system as APT malware and unable to determine whether these suspicious samples are related to the APT attacks [7]. Additionally, manual analysis of samples at present is impractical. When an intrusion detection system detects suspicious samples and issues an alert, it calls network security experts for a long time to carry out manual analysis to determine whether the samples belong to particular APT attacks [8]. Due to the excessive number of alarms, network security experts have brought great pressure. Therefore, as APT attacks become more frequent today, accurately identifying APT malware from suspicious samples has become an imminent problem.

How to distinguish APT malware from ordinary malware? Currently, there are two methods for solving this problem: dynamic and static analyses [9]. It means training the classification model by extracting dynamic or static features [2, 3, 7, 8, 10, 11]. Static analysis technology can inspect executable programs without actual execution samples; the main advantage it has is that it is not affected by execution expenses, whereas dynamic analysis is to observe executable programs in real or virtual execution environments and monitor actual malicious acts [12]. However, the time and resources consumed by dynamic analysis will be a significant disadvantage compared to static analysis; it is unclear what environmental APT malware needs and the time it requires to observe [7]. Additionally, some researchers have combined static and dynamic analyses to conduct a mixed analysis [2]. The aim is to employ the advantages of these two analytical techniques. In order to make a relatively reasonable explanation of the results, compared with other works, this work can quantify the influence and weight of a feature on classification. The TF-IDF method was used in previous studies to determine the priority and weight of features. Moreover, TF-IDF is a method based on people's prior knowledge. It has cognitive bias; thus, how to improve the priority calculation method is worth studying.

The threat intelligence reports published on the network recently confirmed that APT attacks have a high reuse rate. Each APT group has its characteristics, and the APT of the same APT family has some similarities. Therefore, most APT attacks are variants of existing attacks [8, 13]. The target of APT attackers in the same organization is often similar and regular; thus, the behavior of malware and the target of attack are often regular. Different APT organizations are interested in different targets. For example, APT33 has only been found to target the transport system of Saudi Arabia [13]. Therefore, if the APT malware was found in which family they belong to, it can be searched for the APT threat intelligence report information of this family published on the network and investigate their characteristics. Then, according to the characteristics of APT, we can take active and targeted defensive measures [2]. Therefore, it is necessary to analyze the information of malware from different APT families. Understanding the malicious behaviors of different APT families can enhance understanding and resisting APT attacks [2]. To identify families of APT malware samples, the current work analyzes typical malicious malware behaviors of different APT families to distinguish them [2, 14]. However, the number of publicly available malware samples from each APT family is small, making it difficult to train a robust classification model through such a small number of samples [15].

The study aims to solve the above problem and improve existing methods by employing the knowledge in the natural language processing (NLP) field. We also proposed an APT malware classification method by combining multiple depth learning algorithms and transfer learning, since the system call application programming interface (API) information has a severe effect on malware detection [16], and it supports a higher level of samples behavior analysis [12, 17]. Therefore, we selected API as the feature by calculating the priority and contribution degree of each feature. The function of calculating the probability that the test sample belongs to each APT family is realized. According to the obtained results and threat intelligence report, targeted defense and detection can be conducted for specific APT attacks. Finally, we trained and tested APT malware and ordinary malware samples collected from the network and successfully tested the accuracy of the classification method.

Thus, the contributions of this study are as follows:

(1) The binary classification task of the APT and ordinary malware is completed by fusing multiple deep learning algorithms, making the model more active for training and achieving better results than similar studies.

(2) Through the deep learning algorithm, the classification contribution and weight of system call features are dynamically calculated to obtain the priority and probability of different APT attacks, making the model more interpretable and convincing.

(3) Transfer learning is used to transfer the training results of binary classification to the multi-classification task of the APT family to make the model converge faster and strengthen the generalization ability. Thus, the problem of the small number of malware samples in APT is solved.

The remainder of the paper is organized as follows: Section 2 introduces related studies. Section 3 provides an overview of the framework. Section 4 explains the proposed methodology. In Section 5, we introduce the datasets. Section 6 presents the experimental and comparative test results to verify the effectiveness of the proposed method. Finally, Section 7 presents the conclusion.

2. Related Work

Research on common malware detection technology can be divided into static-feature-based and dynamic-feature-based methods. The software run time behavior is one of the dynamic features. However, the dynamic-feature-based methods discussed in most studies are behavior-based [18].

Studies based on static features are as follows: Kang and Won [19] proposed a method to extract feature data from files and detect malware using machine learning. They constructed a malware classification model using multiple

DNN, XGBoost, and Random Forest layers. They also analyzed its performance and obtained an accuracy of 96.3%. Li et al. [15] proposed an incremental malware classification (IMC) framework and an incremental learning method based on multiclass support vector machines (SVM), which improved the classification ability of IMCSVM incrementally by learning new malware samples. Baldangombo et al. [20] presented a static malware detection system to extract valuable features of Windows portable executable (PE) files using the static analysis method.

Behavior-based studies are as follows. Lin et al. [21] developed an SVM classifier for malware classification based on the behaviors of malicious software collected in a sandbox environment. This method combines feature selection and extraction and significantly reduces the feature dimension. It is used for training and classification. Mohaisen et al. [22] proposed a behavior-based automatic malware analysis and marking (AMAL) system to monitor the use of malware on the file system, memory, network, and registry. It creates representative features according to the above situation and uses them to build a classifier trained by manually reviewed training samples. These classifiers can classify malware samples into families with similar behavior. Alazab et al. [23] proposed a method to extract and analyze the characteristics of API calls automatically, attempting to analyze and classify the behavior of API function calls according to the malicious intent hidden in any package program automatically. Amer and Zelinka [24] presented a Markov chain-based method for malware detection. They introduced word embedding to understand the contextual relationships between API functions in the malware call sequence. Gianni et al. [25] proposed a new algorithm based on repeated subsequence alignment, which uses association rules to infer malware behavior. This method takes advantage of the probability of conversion of two API calls in the call sequence. It can operate in the dynamic analysis scenario of tracking API calls at run time.

Currently, the performance of deep learning has attracted significant attention in the network security field [26]. Different from the above machine learning methods, studies on detection using deep learning methods are as follows. Hou et al. [27] proposed a dynamic analysis method based on the component traversal. They used the deep learning framework based on graph features to detect newly unknown Android malware by constructing weighted directed graphs. Hardy et al. [28] investigated how to design a deep learning architecture based on the stackable autoencoder (SAE) model for intelligent malware detection based on the extraction of Windows API calls from PE files. The experimental results show that, compared with the traditional shallow learning method, this method can further improve the overall performance of malware detection. Kolosnjaji et al. [29] constructed a neural network based on the convolution and cyclic network layer to obtain the best features for malware classification. Schofield et al. [30] proposed a convolutional neural network based on Windows system API calls for malware type classification. Kolosnjaji et al. [31] implemented a neural network consisting of convolution and feedforward neural structures, representing a layered feature extraction method that combines the convolution of instruction sequences with pure vectorization of features from PE file headers.

Reading the above work on detecting ordinary malware can bring many suggestions and inspiration to the work in the APT field. Studies on APT detection are as follows. Milajerdi et al. [32] presented the Holmes system and designed a high-level chart to effectively utilize the correlation between suspicious information flows generated during the attacker's activities for APT detection. However, this method requires strong prior knowledge, and it is not easy to deal with complex and variable attacks in APT. Han et al. [33] presented the UNICORN system based on the graph method. They summarized the execution of the long-running system with space efficiency to counter the slow-motion attack of APT occurring in a long period. The disadvantage is that the attacker might poison the dynamic modeling at run time. Cao [34] presented a framework for APT detection, PULSAR. PULSAR uses a probability-graphic model to infer the temporal evolution of attacks based on the security events observed at run time. Ghafir et al. [35] proposed the MLAPT system based on machine learning, which can accurately and quickly detect APT attacks through alarm correlation. However, its limitation lies in overreliance on the accuracy of alarms. Narayanan et al. [36] described a novel cognitive network security system that takes information from different textual sources and stores it in a common knowledge graph using terms from an extended version of a unified network security ontology. Then, the system deduces the knowledge graph of various cooperative agents representing the host and network-based sensors to reduce the load of the security administrator. However, due to confidentiality reasons, APT network threat intelligence from various sources is generally difficult to obtain [2].

Combine malware detection with APT detection: Han et al. [2] designed a new APT malware detection and cognition framework, APTMalInsight, to identify and recognize APT malware using a system call information and ontology knowledge. They proposed an APT malware detection method based on dynamic behavior characteristics. First, the dynamic API sequence is extracted from APT malware. Second, it calculates the classification contribution of API, and the API sequence is sorted. Finally, the effective detection and family classification of APT malware are realized. However, the method to judge the priority of API is based on prior knowledge, which may lead to a deviation. Laurenza et al. [3] relied on the static characteristics of malware and designed a malware classification framework based on the concept of isolation forest learning. They trained each isolation forest with specific APT samples using only static features. To solve the problem of malware APT organization identification, Chen et al. [4] designed a gene model combined with the knowledge graph of malware behavior. They proposed a genetic similarity algorithm for malware APT organization identification and revealed the possibility of using genes to trace malware. Laurenza et al. [7] designed a sample prioritization method, where the known APT obtained from the public report is used to build a knowledge

base for classification. However, it is limited to using only static features and cannot be analyzed according to the behavior of APT malware. Sexton et al. [11] established a classification based on the similarity between programs and known APT malicious software subroutines, indicating that malicious programs and benign programs can share a large number of codes. They used only opcodes as static features, which have limitations. Martín Liras et al. [8] proposed using the static, dynamic, and network-related characteristics through the domain knowledge interpretation and choice, as well as the well-known machine learning techniques to analyze the discriminability of APT-related malware from generic malware without any known association to APT. However, the machine learning classification algorithm model is not active.

3. Methodology

3.1. Detection Framework. Figure 1 shows our detection framework. The detection framework consists of a data analysis module, binary classification model, and multiclassification model. The detailed information is as follows.

The data analysis module is used to test the sample data for the initial data cleansing and exclude some samples that do not fall within the scope of the classification task, such as benign samples. The function of the dichotomous model is to classify input samples by the trained model, the classification category with ordinary malware, and APT malware two dimensions. Then, the APT malware samples output by the binary classification model are taken as the input of the multiclassification model. The probability of each sample corresponding to each APT family is calculated through the scoring system, making the detection results more intuitive and helping to judge the possibility of various APT attacks. The details of the training model are described below.

3.2. Training Framework. Figure 2 shows the model training framework. The training model process consists of data preprocessing, feature extraction, string segmentation, various deep learning algorithms, classifiers, and transfer learning. The detailed information is shown as follows.

First, the datasets of the common and APT malware are collected from public. After obtaining the datasets, data are cleaned, and the API system call features are extracted. To better express the meaning of the feature, expand the representation space of the feature, and solve the out-of-vocabulary (OOV) problem, we segment the string corresponding to each API of each sample to obtain the embedding of the segmented string. Then, the dynamic long short-term memory (LSTM) algorithm in the NLP domain is used to obtain the embedding of each API. After that, the attention algorithm was used to integrate all API under each sample with weight to obtain the sample representation vector and map the sample representation vector to a two-dimensional space to get a two-dimensional vector. Finally, the softmax binary classification of APT and common malware was performed.

Different from the TF-IDF method used in similar work to examine the priority and weight of features [2, 3, 7, 8, 10, 11], our feature weight is trained based on the attention algorithm. Therefore, which API has a greater impact on the sample classification will be given higher weight by the model. TF-IDF method is a priority determination method based on the prior knowledge of human beings, whose prior knowledge may have cognitive bias. Our method avoids cognitive bias; however, it makes the model more interpretable and authentic by calculating the classification contribution of API.

Based on the binary classification model, we propose a transfer learning method, which embeds different strings in the optimal binary training model as the initial embedding of the corresponding strings in multiple classifications to deal with the softmax multiple classifications of malware in different APT families. However, the model has a fast convergence speed and strong generalization ability. It also solves the problem of a small number of APT malware samples.

The model training framework uses different deep learning algorithms to complete the binary classification tasks of APT malware, common malware, and the multiclassification tasks of the APT malware family to make the model more active for training and calculate the classification contribution and weight of API on this basis. Then, it can obtain the priority and probability of different APT attacks, making the model more interpretable and persuasive.

Figure 3 shows our network structure diagram.

3.3. Data Preprocessing. After obtaining the initial sample data, the first task is to preprocess the original dataset. First, parts of ambiguous data samples are screened manually using the online malware sample analysis function of VirusTotal, which is the world's largest online malicious file analysis website. It allows 75 kinds of antivirus software to identify the target samples and report the results, including almost all antivirus software programs worldwide. Suppose that a sample is analyzed by the VirusTotal website, and only less than three antivirus software programs identify the sample as malware. In that case, we consider the sample to have a high probability of being benign and exclude it from the dataset. If there are errors in the data samples in the original dataset, the model will be troubled during the training. To avoid such a situation, we choose the data preprocessing method to minimize the uncertainties in the dataset.

Additionally, after the feature extraction, there is a step to process the dataset. After completing the feature extraction task, we found that feature sequences were the same as those of some samples, indicating redundant data in the dataset. We would only retain one sample with the same feature sequence. The number of features in some feature sequences is small and unrepresentative; thus, we decide to screen out samples with less than ten features and exclude them from ensuring that the model can be less affected by uncertain factors in the training process.

3.4. Feature Extraction and Processing. First, pefile is used to extract API of the samples; pefile is an open-source project based on Python. It has the advantages of agile development, convenience, and quick access to various key data structures of samples.

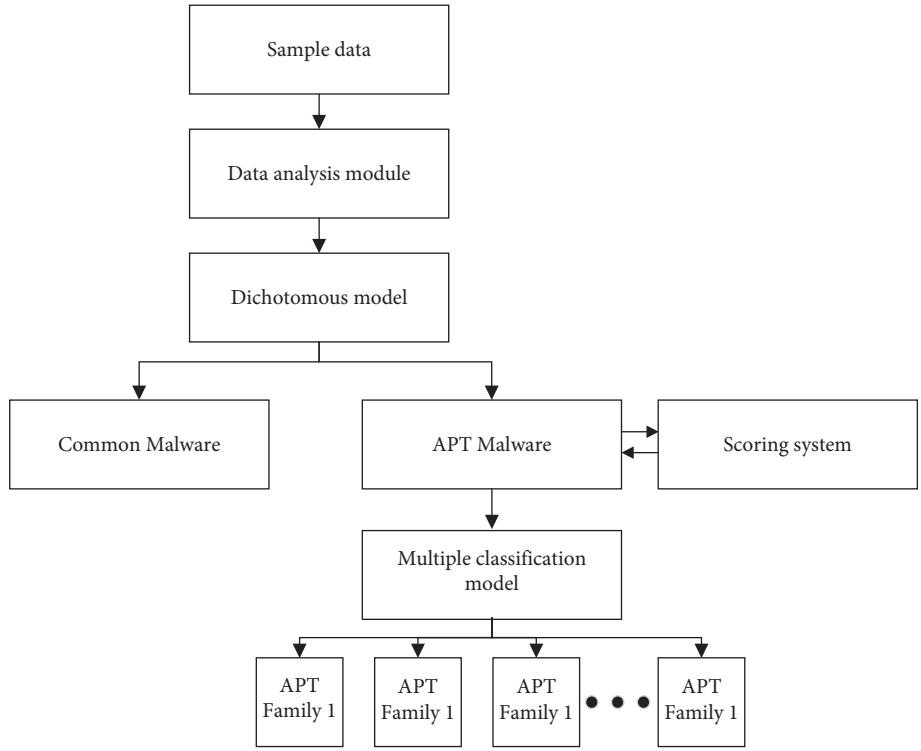

Figure 1: Detection framework. The main function of the detection framework is to detect the input samples.

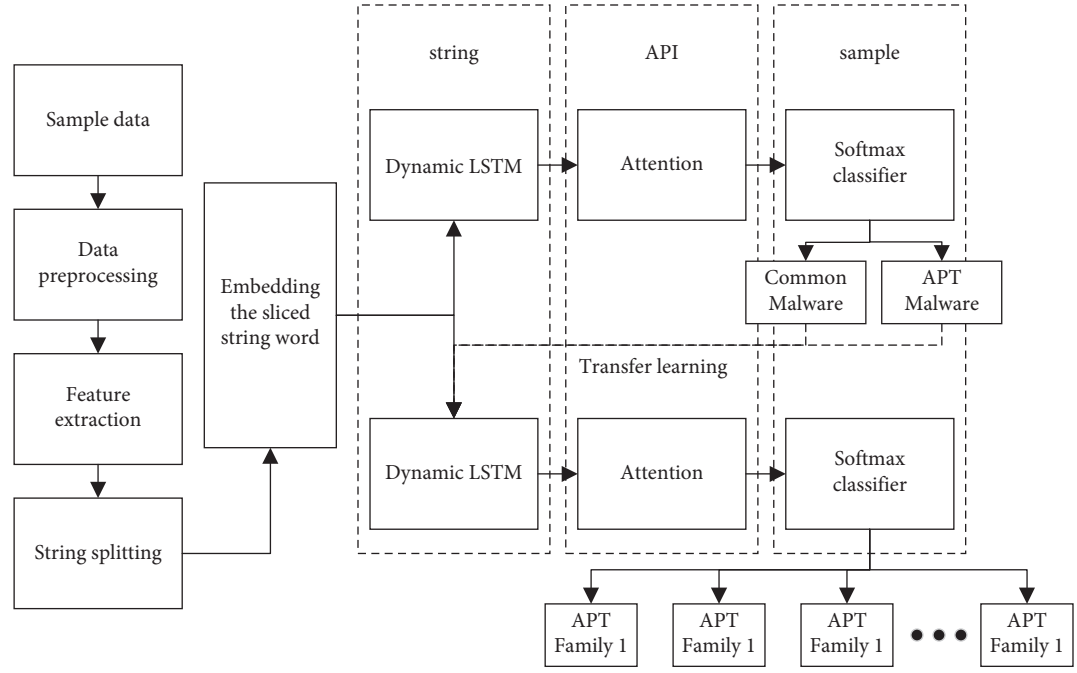

Figure 2: Training framework. The training framework describes the training process and specific steps of the classification model.

We selected API as the characteristics of the samples because attackers can directly, with the native operating system application API, interact to perform a behavior. The native API provides a kernel, calling the underlying operating system services controlled, such as those related to hardware, devices, memory, and service processes. The operating system uses these native APIs during boot (when other system components have not been initialized) and performs tasks and requests during normal operations. The functionality provided by native APIs is often exposed to user-mode applications through interfaces and libraries. Higher-level software frameworks (such as Microsoft .NET and MacOS Cocoa) can be used to interact with native APIs. Attackers may abuse these native API functions as a means

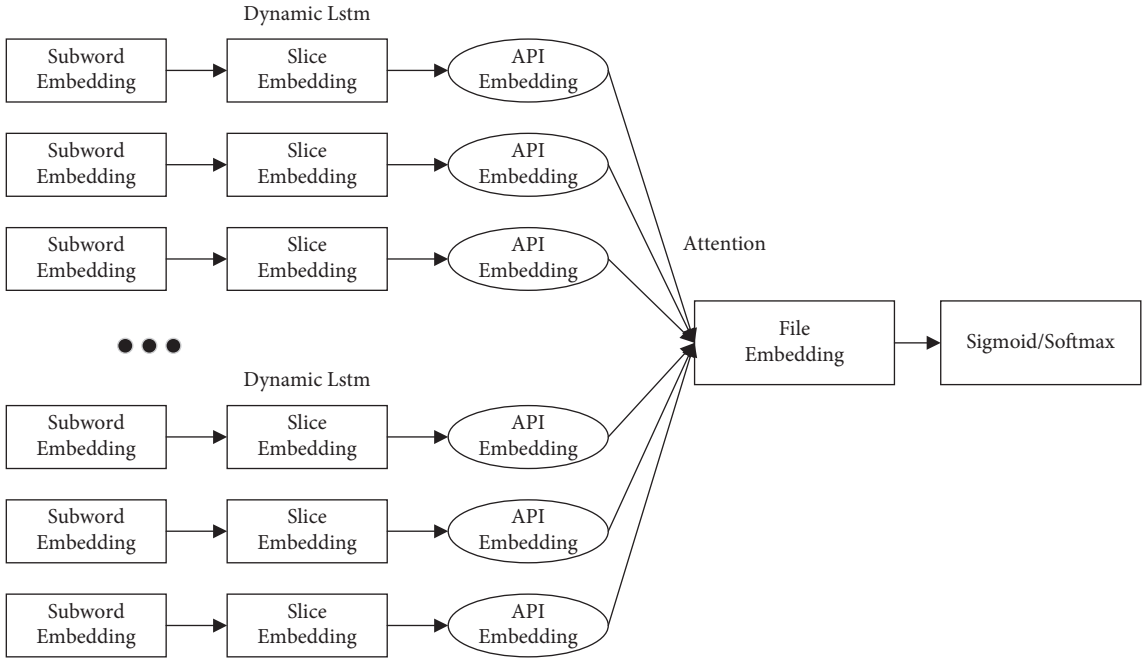

Figure 3: Network structure diagram. The deep learning network structure combining dynamic LSTM and attention algorithm is used to train the classification model.

of performing a behavior. Similar to command and script interpreters, the native API and its interface hierarchy provide a mechanism for interacting with and exploiting the various components of the victim system. Thus, it can be monitored to identify adversary activity.

To better express the meaning of API and solve the OOV problem, we performed string segmentation on API and assigned the same embedding to the same string after segmentation (Figure 4).

When performing NLP or text processing, we usually have a vocabulary. The vocabulary is either preloaded, self-defined, or extracted from the current dataset. Suppose that we have another dataset with words that are not in our current vocabulary. Let us say these words are OOV. In this study, if an API feature of a sample in the test set does not appear in the vocabulary of the training data, this problem will be largely avoided using our approach.

3.5. Classification Algorithm. Embedding converts large sparse vectors in a low-dimensional space that retains semantic relations, that is, to find a mapping or function to generate an expression in a new space. Positions (distance and direction) in the vector space can code the semantics into a good embedding. In this study, we use a Gaussian distribution to initialize the embedding randomly. The reason is that a large amount of data in this study is sparse, and the embedding can be well analyzed and understood through embedding.

LSTM is an RNN used to solve the gradient disappearance and explosion problems in the long sequence training process. For simplicity, LSTM can perform better in longer sequences and better understand contextual semantic information and contextual relationship information than ordinary RNN. Thus, it has a better fit in the experimental environment of this study.

RNN has only one transmission state, whereas LSTM neurons transmit two pieces of information backward in time dimension: one cell state and one hidden state. The hidden state stores mostly "recent memory." What is stored in the cell state is mainly "long-term memory." Figure 5 shows the structure of LSTM.

There are three main phases in the LSTM:

(1) Forget phase. This phase is about selectively forgetting the input from the previous node implemented using the Sigmoid layer known as the "forget gate." It looks at previous output and current input and prints out a number between 0 and 1 for each number in the cell state (previous state), with 1 representing full retention and 0 representing total deletion.

$$f_t = \sigma\left(W_f \cdot \left[h_t = 1, x_t\right] + b_f\right). \tag{1}$$

(2) Choose the memory phase. The input of this stage is selectively "memorized." First, the Sigmoid layer, known as the "input gate layer," determines which values we will update. Next, a tanh layer is used to create candidate vectors. Then, the two vectors are combined to create an updated value. Here, $*$ is the new candidate value. In this study, we add the embedding of a new string to the cell state to replace the old object forgotten in the previous step by updating the last state value with the following formula:

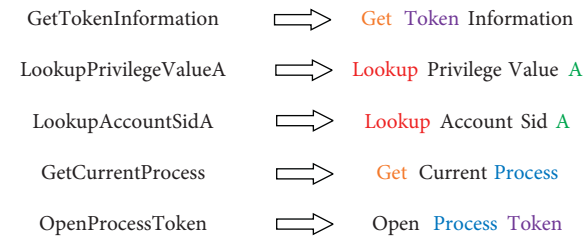

Figure 4: Segmentation of the API system call information. Words with the same color in the figure are given the same embedding.

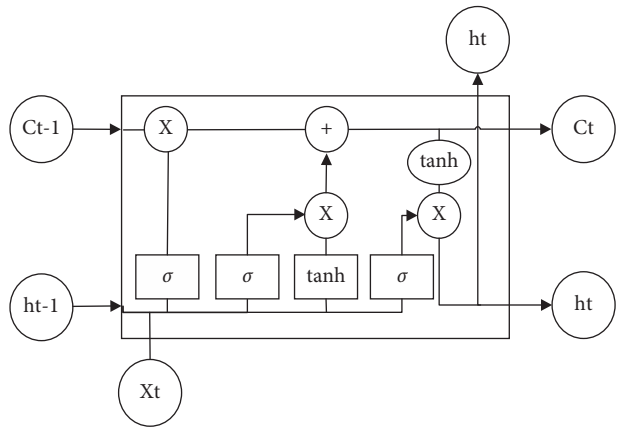

Figure 5: Interactive neural network layer in LSTM. LSTM can better solve the problems of gradient disappearance and gradient explosion in the long sequence training process.

$$\begin{aligned} i_t &= \sigma\left(W_i \cdot \left[h_t - 1, x_t\right] + b_i\right), \\ \widetilde{c}_t &= \tanh\left(W_c \cdot \left[h_{t-1}, x_t\right] + b_c\right), \\ c_t &= f_t * c_{t-1} + i_t * \widetilde{c}_t. \end{aligned} \tag{2}$$

(3) Output phase. This phase determines what will be treated as the output of the current state, first by a Sigmoid layer, to determine which parts to output, and then by passing the cell state through the tanh layer and multiplying it by the Sigmoid layer's output:

$$\begin{aligned} o_t &= \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right), \\ h_t &= o_t * \ \tanh\left(c_t\right). \end{aligned} \tag{3}$$

Compared with ordinary LSTM in this experimental environment, dynamic LSTM combines the sequence and semantic information of the string segmented by API into a vector with fixed dimension size. However, if ordinary LSTM is used, the calculation will be redundant, and the filled zeros will be repeatedly calculated, leading to errors in the results.

The attention mechanism is a method for solving a problem by imitating human attention. Simply put, the attention mechanism is to quickly screen high-value information from a large amount of information. It is mainly used to solve the problem that it is difficult to obtain the final reasonable vector representation when the input sequence of the LSTM/RNN model is long. The method is to retain the intermediate results of LSTM, learn them with the new model, and correlate them with the output to achieve information screening.

The attention principle is to calculate the degree of match between the current input sequence and output vector. The higher the degree of match, the higher the point of attention and the higher the relative score. The match weight calculated from attention is limited only to the current sequence pair, not the overall weight, such as the weight of the network model. We used attention to combine all APIs under each file with weights to obtain the representation vector of each sample.

Our attention function is the dot-product attention. The essence of the dot-product attention is to address the operation process (Figure 5). There are three vectors, Q, K, and V, representing Query, Key, and Value, respectively. They are obtained from different linear transformations of the word embedding result.

$$\begin{aligned} \text{query} &= \text{self} \cdot \text{linear\_query}\left(\text{query}\right), \\ \text{key} &= \text{self} \cdot \text{linear\_keys}\left(\text{query}\right), \\ \text{value} &= \text{self} \cdot \text{linear\_values}\left(\text{query}\right). \end{aligned} \tag{4}$$

Given a task-related Query vector, Value can be calculated by calculating the attention distribution of Key and appending it to Value. This is a manifestation process of the attention mechanism alleviating the complexity of the neural network model. Figure 5 shows the mechanism of attention algorithm (Figure 6).

Perform all API attention simultaneously.

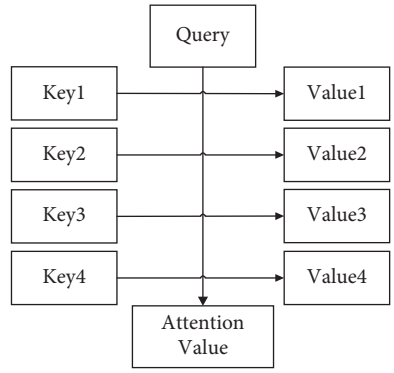

Figure 6: Overview of the mechanism of attention algorithm.

$$\text{Attention}(Q,K,V) = \text{soft max}\left(\tanh\left(\begin{bmatrix} v_1 \\ v_2 \\ \dots \\ v_n \end{bmatrix} * \left[v_1^T, v_2^T, \dots, v_n^T\right]\right)\right) * \begin{bmatrix} v_1 \\ v_2 \\ \dots \\ v_n \end{bmatrix} = \text{soft max}\left(\tanh\left(QK^T\right)\right)V. \quad (5)$$

To avoid embedding repeated training features and speeding up the convergence of multiple classification models, we use the transfer learning method to generalize the models better. Transfer learning transfers the parameters of the trained model (pretrained model) to the new model to help the training of the new model. Considering that most data or tasks are correlated, we have a strong correlation between the characteristics of dichotomies and multiclassification models. Therefore, through the transfer learning, we can share the model parameters learned with the new model to speed up and optimize the learning efficiency of the model instead of learning from zero, as most networks do. It also solves the problems of the small number of samples, small number of features, and difficult fitting for multiclassification.

In this study, the isomorphic inductive transfer learning method is adopted to solve the problem of different learning tasks in the source domain and target domain, as well as the problem of the same feature dimension and different distribution.

4. Datasets

We collected datasets from public. We collected 10841 samples of common malware (https://github.com/iosifache/DikeDataset) belonging to the APT1, APT10, APT19, APT21, APT28, APT29, APT30, Dark Hotel, Energetic Bear, Gorgon Group, WinNTI, and other APT organizations and groups of 11 APT malware samples (https://github.com/cyber-research/APTMalware) and 3,954 APT malware samples. APT malware's provider uses open-source threat intelligence reports from multiple vendors. A number of threat intelligence reports were collected, and a hash list of all files was used as intrusion indicators (IoCs) to obtain target samples from VirusTotal. Table 1 shows the APT malware family and sample size.

5. Experimental Results and Discussion

The whole of the code required for our experiment was written using the Python PyTorch framework. The experimental environment was Windows 10 operating system, Intel(R) Core(TM) i7-4720HQ 2.60 GHz processor, 16 GB RAM, and GTX970M graphics card. The data used in the experiment were 14795 samples of common malware and APT malware samples. For comparison with similar studies, this experiment uses tenfold cross-validation to evaluate the effect of the model. The ratio of the training set to the validation set is 9 : 1. The training time of onefold is 4 hours and 30 minutes on average.

5.1. Evaluation Indexes. First, we evaluate the effectiveness of the binary classification model in classifying APT malware from common malware. Then, we evaluate the results of the APT family classification of the multiclassification model. In evaluating the dichotomy model, positive and negative samples are unbalanced; thus, the accuracy rate cannot be used to evaluate the model. Here, we use the precision rate, recall rate, and $F1$-score to evaluate the model. These evaluation indexes are defined as follows.

Here, TP is the number of positive samples predicted to be positive, TN is the number of negative samples predicted to be negative, FP is the number of negative samples predicted to be positive, and FN is the number of positive samples predicted to be negative.

5.2. Analysis of Experimental Results. We use the tenfold cross-validation method to evaluate the declassification model. Each folding training is 50 rounds, a total of tenfold. Then, the average value is taken to obtain the accuracy rate, precision rate, recall rate, and $F1$-score of the model. The training process is visualized using the TensorBoard

Table 1: APT malware family and sample quantity list.

Sample name	Sample size
APT1	405
APT10	244
APT19	32
APT21	106
APT28	214
APT29	281
APT30	164
Dark Hotel	273
Energetic Bear	132
Gorgon Group	961
WinNTI	387

visualization tool. The accuracy, precision, recall, and *F*1-score are 0.99224, 0.98076, 0.98152, and 0.9811, respectively. The training process is shown in Figures 7–10.

The training accuracy of the multiple classification models of the APT malware family is shown in Figure 11.

This study can also sort the API family of the test samples and calculate the importance of the API in the sample for classification and the probability of classification into each APT family. As shown in Figure 12, we choose a sample of APT21 for testing. We can observe the top 20 most important APIs that the model considers affecting the classification of this sample. The sample was judged to have a 99.94% probability of belonging to APT21. Therefore, the proposed model can effectively determine the ownership of malware.

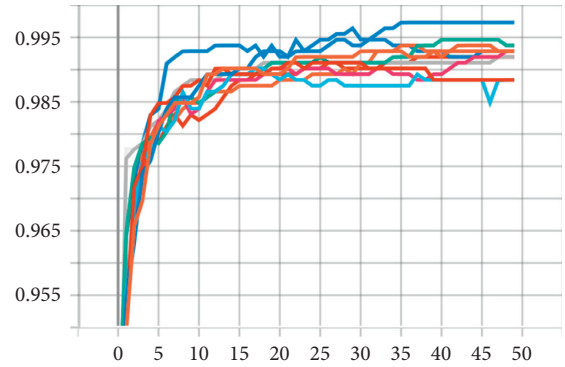

Figure 7: Accuracy of dichotomous model. The abscissa represents the number of iterations, and the ordinate represents the value of accuracy.

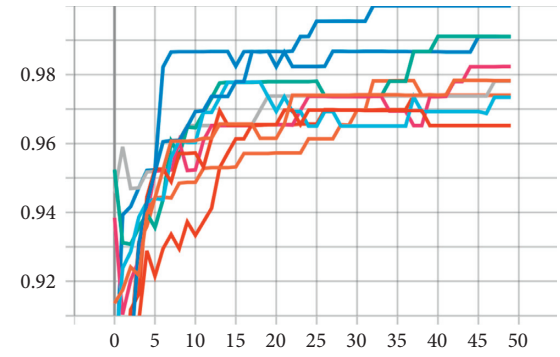

Figure 8: Precision of dichotomous models. The abscissa represents the number of iterations, and the ordinate represents the value of precision.

5.3. Contrast Experiment. First, classical machine learning classification algorithms are used to test the dataset. The test results are as follows. The training results of the proposed models on this dataset are better than the following machine learning algorithms: KNN, logistic regression, decision tree, gradient lifting, AdaBoost, Naive Bayes, linear discriminant analysis, quadratic discriminant analysis, SVM, and polynomial Bayes. We used ten machine learning algorithms, among which gradient lifting and decision tree achieved better results than other algorithms. However, the accuracy of the gradient lifting algorithm is low, only 0.9282, and the recall rate of the decision tree is only 0.9079. Additionally, we test the above algorithm on the multiclassification datasets, and the effect is generally poor. Table 2 shows the Binary classification evaluation results of classical machine learning algorithms.

To better illustrate the influence of the attention method on the experimental results, we compared the proposed model with the method without attention. We obtained that the recall rate and *F*1 value of the model were lower without attention (Figure 13).

This study will also use more classification models using the transfer learning method to compare. As shown in Figure 14, if you do not use the transfer learning model, accuracy will drop by 3% to 4%, and the training time will be longer. In this contrast experiment, the precision, recall rate, and F1-score are not weighted average values.

Compared with the research by Han et al. [2], their research is also to extract the API characteristics of samples. The difference is that they used TF-IDF algorithm to calculate the weight value of each API and sort the API sequence according to the classification contribution and selected only some APIs for model training. However, we use dynamic LSTM and attention to sort the APIs. Besides, they adopted a Random Forest classification algorithm to classify ordinary malware and APT malware samples. We reproduced their method and conducted comparative experiments on their method using our dataset. Figure 15 shows the comparative experimental results. As shown in the figure, our method is better than the method of

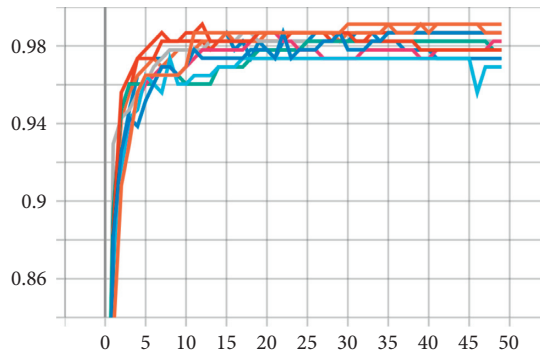

Figure 9: Recall rates for dichotomous models. The abscissa represents the number of iterations, and the ordinate represents the value of recall rates.

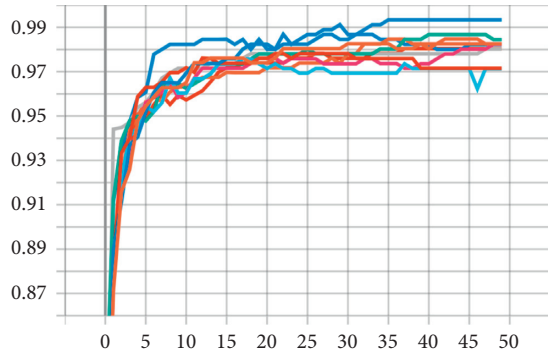

Figure 10: $F1$-score of the dichotomous model. The abscissa represents the number of iterations, and the ordinate represents the $F1$-score value.

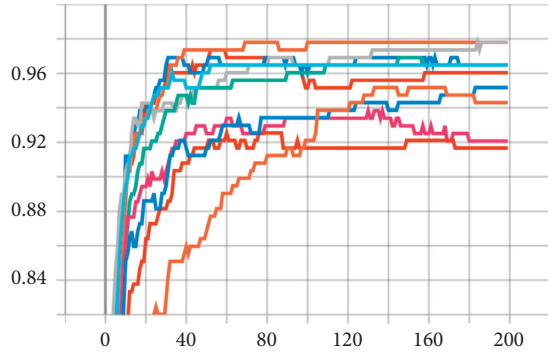

Figure 11: Accuracy of multiple classification models. The abscissa represents the number of iterations, and the ordinate represents the value of accuracy.

Table 2: Binary classification evaluation results of classical machine learning algorithms.

Classification algorithm	Accuracy	Precision	Recall	*F*1
KNN	0.9455	0.9268	0.8333	0.8776
Logistic regression	0.9214	0.8097	0.8026	0.8062
Decision tree	0.9679	0.9324	0.9079	0.92
Gradient boosting	0.9214	0.986	0.9298	0.9571
AdaBoost	0.9282	0.9061	0.7193	0.802
Gaussian NB	0.8045	0.6364	0.2456	0.3544
Linear discriminant analysis	0.9179	0.925	0.6491	0.7629
Quadratic discriminant	0.9161	0.9653	0.6096	0.7473
SVC	0.9473	0.9617	0.7719	0.8564
Multinomial NB	0.3187	0.2102	0.8684	0.3385

```
([{'_XcptFilter': 0.17253531515598297,
  '__getmainargs': 0.027218995615839958,
  '_initterm': 0.02672221139073372,
  '_exit': 0.02669285051524639,
  'CloseHandle': 0.026676882058382034,
  'fread': 0.026674922555685043,
  '_acmdln': 0.02667277306318283,
  'CreateFileA': 0.02667262777686119,
  'GetStartupInfoA': 0.026672562584280968,
  '_adjust_fdiv': 0.026672475039958954,
  '_except_handler3': 0.026672251522541046,
  '__p__commode': 0.026672232896089554,
  'GetSystemInfo': 0.026672225445508957,
  'GetProcAddress': 0.02667219750583172,
  'Sleep': 0.02667219191789627,
  'GetModuleHandleA': 0.026672188192605972,
  'GetVersionExA': 0.026672188192605972,
  'LoadLibraryA': 0.026672188192605972,
  '_controlfp': 0.026672188192605972,
  'memset': 0.026672188192605972},
 {'APT21': 0.9994,
  'APT19': 0.0005,
  'APT30': 0.0001,
  'APT10': 0.0,
  'APT1': 0.0,
  'APT29': 0.0,
  'GorgonGroup': 0.0,
  'APT28': 0.0,
  'Winnti': 0.0,
  'EnergeticBear': 0.0,
  'DarkHotel': 0.0})
```

Figure 12: Feature weight display and classification probability. Top 20 most important APIs affecting sample classification are provided, and the prediction results are also provided.

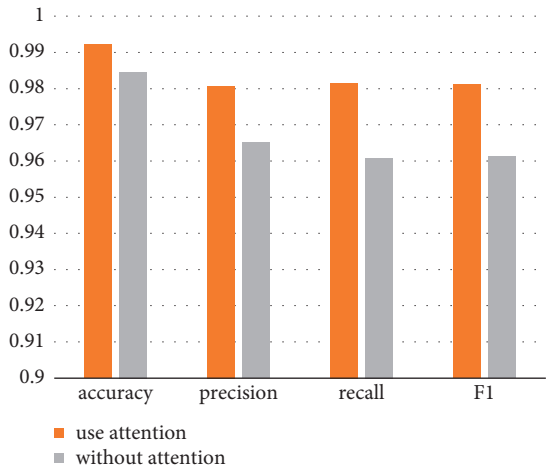

Figure 13: Whether the model uses the contrast of attention.

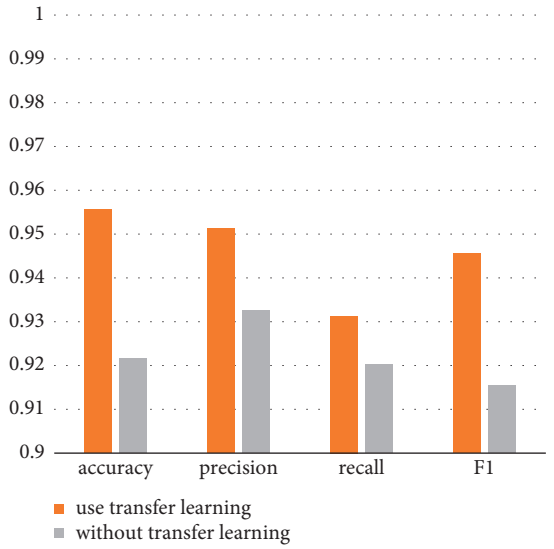

Figure 14: Whether to use transfer learning comparison for multiple classifications.

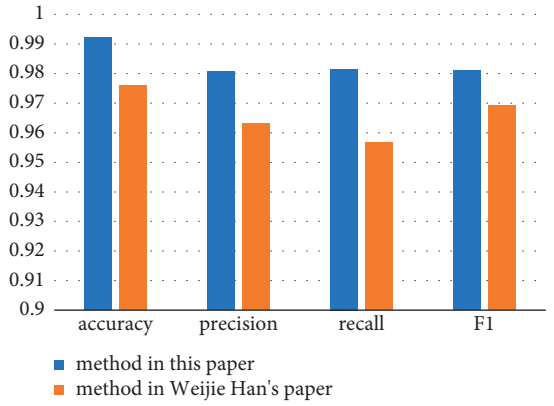

Figure 15: Contrast experiment with Han et al.'s method.

Han et al. This is because the TF-IDF algorithm examines the priority or weight of features based on prior human knowledge, and we pay more attention to the impact of data on model classification.

The above experiments show that the proposed method can effectively solve the identification problem of common malware, APT malware, and the ownership problem of the APT family, demonstrating that there are similarities and connections between the attack means of different APT families.

6. Conclusions

To detect and defend against APT attacks and explore the similarity and connection of attack means of different APT families, this study proposes an APT malware classification method by combining multiple deep learning algorithms and transfer learnings. The experiment indicates that the proposed method can achieve 99.2% in distinguishing common malware from APT malware and assign APT malware to different APT families at an accuracy of 95.5%. The experimental results show that the proposed method is helpful for the classification of the APT malware of different families.

A key issue in the malware ecosystem is its fast evolution and various problems caused by the evolution [37]; sustainability is an important requirement and performance metric; without addressing sustainability, proposing malware detectors/classifiers is an endless task lacking substantial scientific advancement; and the model of this article needs to be continuously updated, continuously collecting APT malware samples for iteration, because APT attacks have strong reusability. Whenever new APT malware is collected, the updated model can effectively defend against such attacks for a period of time in the future. This reflects the lack of continuity, because iteration cannot be carried out automatically, and there is no guarantee that the model will not be affected by adversarial examples. How to ensure the sustainability of the model is our future research direction.

Data Availability

The APT malware data in this paper can be obtained free of charge from https://github.com/cyber-research/APTMalware. The common malware data in this paper can be obtained free of charge from https://github.com/iosifache/DikeDataset.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by the National Natural Science Foundation of China under Grant nos. 61772229 and 62072208 and International Science and Technology Cooperation Projects of Jilin Province under Grant no. 20210402082GH.

References

[1] A. Rot and B. Olszewski, "Advanced persistent threats attacks in cyberspace. Threats, vulnerabilities, methods of protection," in *Proceedings of the Position Papers of the 2017 Federated Conference on Computer Science and Information Systems*, Prague, Czech Republic, September 2017.

[2] W. Han, J. Xue, Y. Wang, Z. Liu, and Z. Kong, "MalInsight: a systematic profiling based malware detection framework," *Journal of Network and Computer Applications*, vol. 125, no. 12, pp. 236–250, Article ID e3884, 2019.

[3] G. Laurenza, R. Lazzeretti, and L. Mazzotti, "Malware triage for early identification of Advanced Persistent Threat activities," *Digital Threats: Research and Practice*, vol. 1, no. 3, pp. 1–17, 2020.

[4] W. ChenX. Helu et al., "Advanced persistent threat organization identification based on software gene of malware," *Transactions on Emerging Telecommunications Technologies*, vol. 31, no. 12, Article ID e3884, 2020.

[5] F. Li, A. Lai, and D. Ddl, "Evidence of Advanced Persistent Threat: a case study of malware for political espionage," in *Proceedings of the 2011 6th International Conference on Malicious and Unwanted Software*, pp. 102–109, Fajardo, PR, USA, October 2011.

[6] G. Zhao, K. Xu, L. Xu, and B. Wu, "Detecting APT malware infections based on malicious DNS and traffic analysis," *IEEE Access*, vol. 3, pp. 1132–1142, 2015.

[7] G. Laurenza, L. Aniello, R. Lazzeretti, and R. Baldoni, "Malware triage based on static features and public apt reports," in *Proceedings of the International Conference on Cyber Security Cryptography and Machine Learning*, pp. 288–305, Springer, Beer-Sheva, Israel, June 2017.

[8] L. F. Martín Liras, A. R. de Soto, and M. A. Prada, "Feature analysis for data-driven APT-related malware discrimination," *Computers & Security*, vol. 104, no. 1, p. 102202, 2021.

[9] N. Udayakumar, S. Anandaselvi, and T. Subbulakshmi, "Dynamic malware analysis using machine learning algorithm," in *Proceedings of the 2017 International Conference on Intelligent Sustainable Systems (ICISS)*, pp. 795–800, IEEE, Palladam, India, December 2017.

[10] M. Latah, *When Deep Learning Meets Security*, 2018, https://arxiv.org/abs/1807.04739.

[11] J. Sexton, C. Storlie, and B. Anderson, "Subroutine based detection of APT malware," *Journal of Computer Virology & Hacking Techniques*, vol. 12, no. 4, pp. 1–9, 2015.

[12] D. Ucci, L. Aniello, and R. Baldoni, "Survey of machine learning techniques for malware analysis," *Computers & Security*, vol. 81, no. MAR., pp. 123–147, 2019.

[13] ThaiCERT, "Threat group cards: a threat actor encyclopedia," *TLP: WHITE Version 2. 0*, pp. 48-49, 2020, https://www.thaicert.or.th/downloads/files/Threat_Group_Cards_v2.0.pdf.

[14] I. Rosenberg, G. Sicard, and E. David, "End-to-End deep neural networks and transfer learning for automatic analysis of nation-state malware," *Entropy*, vol. 20, no. 5, p. 390, 2018.

[15] J. Li, D. Xue, W. Wu, and J. Wang, "Incremental learning for malware classification in small datasets," *Security and Communication Networks*, vol. 2020, no. 20, 12 pages, Article ID 6309243, 2020.

[16] M. Belaoued and S. Mazouzi, "Statistical study of imported APIs by PE type malware," in *Proceedings of the 2014 International Conference on Advanced Networking Distributed Systems and Applications*, pp. 82–86, Bejaia, Algeria, June 2014.

[17] F. O. Catak and A. F. Yazi, *A Benchmark API Call Dataset for Windows PE Malware Classification* 2019, https://arxiv.org/abs/1905.01999.

[18] M. Al-Kasassbeh, S. Mohammed, M. Alauthman, and A. Almomani, "Feature selection using a machine learning to classify a malware," *Handbook of Computer Networks and Cyber Security*, Springer, Cham, pp. 889–904, 2020.

[19] J. Kang and Y. Won, "A study on variant malware detection techniques using static and dynamic features," *Journal of Information Processing Systems*, vol. 16, no. 4, pp. 882–895, 2020.

[20] U. Baldangombo, N. Jambaljav, and S. J. Horng, "A static malware detection system using data mining methods," *International Journal of Artificial Intelligence & Applications*, vol. 4, no. 4, 2013.

[21] C. T. Lin, N. J. Wang, H. Xiao, and C. Eckert, "Feature selection and extraction for malware classification," *Journal of Information Science and Engineering*, vol. 31, no. 3, pp. 965–992, 2015.

[22] A. Mohaisen, O. Alrawi, and M. Mohaisen, "AMAL: high-fidelity, behavior-based automated malware analysis and classification," *Computers & Security*, vol. 52, no. Jul, pp. 251–266, 2015.

[23] M. Alazab, S. Venkataraman, and P. Watters, "Towards understanding malware behaviour by the extraction of API calls," in *Proceedings of the 2nd Cybercrime Trustworthy Comput. Workshop*, pp. 52–59, Ballarat, VIC, Australia, July 2010.

[24] E. A. Amer and I. Zelinka, "A dynamic Windows malware detection and prediction method based on contextual understanding of API call sequence," *Computers & Security*, vol. 92, Article ID 101760, 2020.

[25] D. A. Gianni, F. Massimo, and F. Palmieri, "Association rule-based malware classification using common subsequences of API calls," *Applied Soft Computing Journal*, vol. 105, Article ID 107234, 2021.

[26] K. Saba and G. Reena, *A Survey of Cyber Security Operations Based on Machine Learning & Deep Learning," Volume VIII, Issue II*, pp. 168–173, 2019.

[27] S. Hou, A. Saas, L. Chen, and Y. Ye, "Deep4MalDroid: a deep learning framework for android malware detection based on linux kernel system call graphs," in *Proceedings of the 2016 IEEE/WIC/ACM International Conference on Web Intelligence Workshops (WIW)*, pp. 104–111, Omaha, NE, USA, October 2016.

[28] W. Hardy, L. Chen, S. Hou, Y. Ye, and X. Li, "DL4MD: a deep learning framework for intelligent malware detection," in *Proceedings of the International Conference on Data Mining (DMIN), page 61. The Steering Committee of The World*

Congress in Computer Science, Computer Engineering and Applied Computing(WorldComp), Las Vegas, NV, USA, 2016.

[29] B. Kolosnjaji, A. Zarras, G. Webster, and C. Eckert, "Deep learning for classification of malware system call sequences," in *Proceedings of the AI 2016: Advances in Artificial Intelligence*, pp. 137–149, TAS, Australia, December 2016.

[30] M. Schofield, G. Alicioglu, R. Binaco, and P. Turner, "Convolutional neural network for malware classification based on API call sequence," in *Proceedings of the 8th International Conference on Artificial Intelligence and Applications (AIAP 2021)*, Sydney, Australia, January 2021.

[31] B. Kolosnjaji, G. Eraisha, G. Webster, A. Zarras, and C. Eckert, "Empowering convolutional networks for malware classification and analysis," in *Proceedings of the 2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 3838–3845, IEEE, Anchorage, AK, USA, May 2017.

[32] S. M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. N. Venkatakrishnan, "HOLMES: real-time APT detection through correlation of suspicious information flows," in *Proceedings of the 2019 IEEE Symposium on Security and Privacy (SP)*, May 2019.

[33] X. Han, T. Pasquier, A. Bates, J. Mickens, and M. Seltzer, *UNICORN: Runtime Provenance-Based Detector for Advanced Persistent Threats*, 2020, https://arxiv.org/abs/2001.01525.

[34] P. Cao, *On Preempting Advanced Persistent Threats Using Probabilistic Graphical Models*, 2019, https://arxiv.org/abs/1903.08826.

[35] I. Ghafir, M. Hammoudeh, V. Prenosil et al., "Detection of advanced persistent threat using machine-learning correlation analysis," *Future Generation Computer Systems*, vol. 89, no. DEC, pp. 349–359, 2018.

[36] S. N. Narayanan, A. Ganesan, K. Joshi, T. Oates, A. Joshi, and T. Fin, "Early detection of cybersecurity threats using collaborative cognition," in *Proceedings of the 2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC)*, October 2018.

[37] G. Suarez-Tangil and G. Stringhini, *Eight Years of Rider Measurement in the Android Malware Ecosystem: Evolution and Lessons Learned*, 2018, https://arxiv.org/abs/1801.08115.