11-17-2020

A Real-Time and Adaptive-Learning Malware Detection Method Based on API-Pair Graph

Shaojie Yang

Shanxi Li

Wenbo Chen

Yuhong Liu
Santa Clara University, yhliu@scu.edu

Recommended Citation

Yang, S., Li, S., Chen, W., & Liu, Y. (2020). A Real-Time and Adaptive-Learning Malware Detection Method Based on API-Pair Graph. IEEE Access, 8, 208120–208135. https://doi.org/10.1109/ACCESS.2020.3038453

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 28, 2020, accepted November 11, 2020, date of publication November 17, 2020,
date of current version November 30, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3038453

A Real-Time and Adaptive-Learning Malware Detection Method Based on API-Pair Graph

SHAOJIE YANG[1], SHANXI LI[1], (Member, IEEE), WENBO CHEN[1], AND YUHONG LIU[2], (Member, IEEE)

[1]School of Information Science and Engineering, Lanzhou University, Lanzhou 730000, China
[2]Department of Computer Engineering, Santa Clara University, Santa Clara, CA 95053, USA

Corresponding authors: Shanxi Li (lisx@lzu.edu.cn) and Wenbo Chen (chenwb@lzu.edu.cn)

ABSTRACT The detection of malware have developed for many years, and the appearance of new machine learning and deep learning techniques have improved the effect of detectors. However, most of current researches have focused on the general features of malware and ignored the development of the malware themselves, so that the features could be useless with the time passed as well as the advance of malware techniques. Besides, the detection methods based on machine learning are mainly static detection and analysis, while the study of real-time detection of malware is relatively rare. In this article, we proposed a new model that could detect malware real-time in principle and learn new features adaptively. Firstly, a new data structure of API-Pair was adopted, and the constructed data was trained with Maximum Entropy model, which could satisfy the goal of weighting and adaptive learning. Then a clustering was practised to filter relatively unrelated and confusing features. Moreover, a detector based on Lont Short Term Memory Network (LSTM) was devised to achieve the goal of real-time detection. Finally, a series of experiments were designed to verify our method. The experimental results showed that our model could obtain the highest accuracy of 99.07% in general tests and keep the accuracies above 97% with the development of malware; the results also proved the feasibility of our model in real-time detection through the simulation experiment, and robustness against a typical adversarial attack.

INDEX TERMS Malware detection, adaptive learning, real-time detection, API-pair graph, deep learning.

I. INTRODUCTION

Since the last few decades, malware has become a common threaten for cybersecurity. Malware will be transferred in computers without the permission of their owners and will harm computers, networks as well as steal target information since malware plays a vital role in damages of cyberspace. Defending of malware had almost appeared with the growth of malware, which will also promote anti-detection techniques of malware.

Most solutions for defending malware were based on two main techniques. Static detection, including signature-based detection and heuristic-based detection, has sound effects on detecting known malware. Moreover, dynamic detection, such as behavior-based detection and pattern-checking-based detection, have a specific rate on both known and unknown malicious program [1]. Additionally, deep learning-based detection has become an essential approach in the research of recognizing malware, which would be applied widely in the future.

However, in contrast to the development in malware detection, technologies refer to malware anti-detection that was proposed and used year by year, which even developed far faster than the detection methods. Many techniques are proposed to avoid detection solutions. Encryption is one of the most straightforward approaches for malware authors to camouflage their program [2]. For malware that was encrypted, it consists of two modules to encrypt and decrypt the program. Once encryption is done, the key will be included in the malware for decryption. The aim of encryption is mainly to avoid static detection based on signature match and code analysis. It could also be used to impede related surveys. Obfuscation is another technique that makes a program harder to understand [3]. For malware, the obfuscation is applied to defend both static detection and dynamic detection. Different approaches are proposed to achieve the target. The easiest way is the insertion of junk code, a series of code that has no meaning except deceive detecting solution. The insertion will

The associate editor coordinating the review of this manuscript and approving it for publication was Mervat Adib Bamiah.

not affect the purpose of the program. Other methods include instruction substitute, which means replace some sensitive instruction by others that have similar functions [4]. Finally, even deep learning-based detection has not been applied diffusely; some anti-detection methods are proposed, which focus on attacking deep learning detection models. One of them is to generate adversarial samples to evade detectors which utilized deep learning or machine learning models; the approach could even decrease the accuracy of detection into zero, which means destroy the whole model.

Recently, researchers have shown an increased interest in the detection based on more fundamental signatures, like API and opcode. Some related works focus on dynamic malware detectors based on system-calls [5]. They have analyzed existed malware and generated system call sequences or calls flow graph (CFG). Then they abstract some dependency maps from the orders or graphs with statistics methods and trained with machine-learning tools to finish a behavior-based detector. However, all of those researches had a basis that samples have been executed, and all their behaviors have been recorded, which means some experimental black-box environments like sandbox or virtual-machine are needed. Furthermore, all analyses have to begin after execution finished, which could be a risk for production environments. For instance, if victims have infected with ransomware, the malware could behave caused damages before detector detectors identify recorded all its behaviors and finish the analysis. All these methods could be classified as static detection, which might not satisfy the need for real-time detection.

Besides the development of anti-detection technologies, the development and generation of new types of malware increased incredibly in recent years, especially ransomware [6]. A report shows that ransomware attacks grew by 118% in the first quarter of 2019 [7]. Unfortunately, both static detection and dynamic detection are based on known features, and only after manual analysis for a new type of malware can those approaches identify them. However, ransomware is a type of malware that blackmails victims by locking their data [8]. For the victims, the solution is too late after traditional approaches get features from human analysis results and can detect the ransomware. In this article, we devise a novel real-time malware detector, which can analyze, record, and identify behaviors of malware automatically. Our method based on system calls and focuses more on real-time detection and adaptively update of features. The detector consists of three parts: a weighting model, a cluster for feature filtering, and a real-time malware detector. These systems aim to solve the following problem:

- Which information would be used for detection that could copy with the development of malware?
- How to extract and update features for detection automatically, considering that new types of malware continuously appeared?
- How to implement real-time detection while keeping the performance?
- How to detect unknown malware as well as adversarial samples that aim at evasion of detectors?

Firstly, to process the features conveniently as well as keep more correlation information, we adopted a new structure based on API call sequences and Markov Chains. For each API call in the series, a vector would be mapped with the API to construct a two-dimension pair as a basic unit so that every sequence would be split into a set of tuples with the maintenance of connection information among calls. The transformation would decrease the difficulty of analyzing and provide a condition for real-time detection.

Besides, we take a maximum entropy model to extract features from the pair graphs, training features and achieve adaptive learning. The model attempt to keep all possibility for predicting uncertain results and the prediction should satisfy all constraints, which is suitable for malware detection and new malware prediction.

Moreover, to defend adversarial attacks as well as improve the performance of real-time detection, a clustering algorithm was adopted to split malicious and benign pairs based on the sequence of calls invoked, so that benign features, junk codes, and disguised calls which were used to deceive detectors would be filtered. The essential malicious pairs would be retained, which could increase the accuracy of the detection and decrease the time consumption of the detector.

Finally, a real-time detector was devised based on an improved LSTM structure, the detector could receive features generated in real-time and give the judgment instantly so that malware could be detected before the execution finished, and more loss could be avoided, which is important for ransomware defending.

Experiments show that our approach could detect malware in normal conditions with a highest accuracy of 99.33%. Besides, with time-series development of malware, our model could keep accuracy above 97%. The model also had a certain effect in detecting unknown samples. Moreover, the model was tested with several adversarial samples to validate the robustness of our model against evasion attacks; the result presented that our model could retain the accuracy above 96% when meeting adversarial attacks. Our contribution is listed as follows:

1) We adopted a two-dimension structure of data (API-Pair) to process with API call sequences, which was based on the thought of Markov Chain. The structure could format variable length API call sequences as well as keep the connection characteristics between calls. Then we adopted an effective model to achieve the goal of weighting to features and update of the feature sets automatically to catch up with the development of malware.
2) We proposed a sequence-based clustering approach to remain the flows that are more suspicious and delete the objects that might mislead the detector. Hence, we could improve the effects of the detector and increase robustness against adversarial attacks.

3) We devised a novel behavior-based malware detector for real-time detection. It utilizes sequence-based calls edge flow and optimal weighted model to achieve real-time and unknown malware detection.
4) We had evaluated our detector using real malware and benign samples in the Windows environment. The testing results proved the effectiveness of our model in production conditions.

The rest of this article is organized as follows: Section II would clarify our problems and key points. Afterwards, we would introduce our method in Section IV. The whole design of our model was then presented in Section III, and the experiment details and results of the evaluation were reported in Section V. Moreover, some discussions of our work were presented in Section VI. Finally, we showed some related work in Section VII and concluded our work in Section VIII.

II. ANALYSIS OF THE PROBLEM

Malware is continuously evolved to avoid detection as well as achieve new goals. On one side, some anti-detection methods have been used in malware such as dynamic encryption and decryption to avoid dynamic malware detection solutions. As a result, the detection solution cannot identify malware before they have finished. On the other side, detectors mainly discover malware using known feature databases (such as signatures and suspicious behaviors). However, malware would be updated by changing their sources and implementation mode so that detectors cannot compare them with known features, and they will escape from detectors.

In this article, we proposed a novel malware detection method to solve the problem described before. Our proposed solution utilized a series of approaches and focused on two main points.

A. REAL-TIME DETECTION

Nowadays, dynamic real-time detection generally monitored some sensitive behaviors, file locations, and memory addresses, which had effects over recent decades. However, most techniques have focused on the single features, while execution of malware includes a set of behavior chains so that attackers could just change sensitive behaviors or features with similar ones, and the malware would pass the detection. The approach has been widely used in many anti-detection methods, and detectors could only extend their feature database to solve the problem contemporarily. Moreover, some malware, especially encryption ransomware, would hide their behaviors with hook/injection or dynamic encryption/decryption technique, and key behavior would be performed at the last moment [9]. When detectors discovered exceptions, it is too late to stop malware, and the goals of attackers have been finished.

According to those new threaten for detection, our primitive objective is to define new units for detectors; the units must have successive features and could be detected immediately. API call sequences are a set of API of a program to finish specific functions. Usually, the call sequences of malware have a strong purpose and can be used to analyze its behavior [10]. Though some anti-detection methods would insert some obfuscated API to hide the purpose of malware, its critical calls that refer to its real goals have to be invoked when execution. In a word, the call sequence is an effective prototype for real-time detection.

B. ADAPTIVE UPDATE OF FEATURES

With the development of detection techniques, techniques for anti-detection and a new type of attack also increased, which need experts must analyze new malware and add their features into databases. The problem is, detectors can have abilities to identify new malware only after experts finished their analysis. Before the manual analysis completed, the detectors have no means to prevent new kinds of malware. Another trend is that new types of malware and attack methods grow explosively, which will take too much effort for specialists to study, and detectors will be weak to the tremendous appearance of new malware.

Considering the development of attacks and anti-detection are based on existing methods, learning from current malware and get the ability to detect unknown types of malware is feasible. Therefore, we proposed an adaptive model to identify the malicious level of each behavior of detected programs. Besides, the proposed model also enabled us to learn behaviors and features from new appeared malware to improve the performance of predicting unknown malware.

III. SYSTEM DESIGN OF THE MODEL

The full structure of our model is shown in Figure. 1.

Firstly, a target program would be executed in a monitor environment. When the program invoked API calls, the monitor would record the API and transferred it to the model. Then the model would transform the API to API pair with the previous API call.

After the transformation, the pair would be caught by the Maximum Entropy Model to get the weight of the pair. The weighted pair would then be identified by Sequential Algorithmic Scheme. If the pair was identified as probably malicious, it would be input to the LSTM-based detector. If the result of the detector was malicious, the API pair would be added to API-Pair Graph, and the graph would be trained by the Maximum Entropy Model to update the weight, or the detector would suspend and wait for next Pair input. If the detection process is done, the detector, filter, and the graph would be initialized.

Due to the weight would update with the input of API sequence, the model could update adaptively with the development of malware. Also, because of BSAS algorithm and the characteristics that the weight would be decided by all samples detected before, the model would have the robustness to attacks that focused on deep learning evasion despite the model fully based on deep learning method.

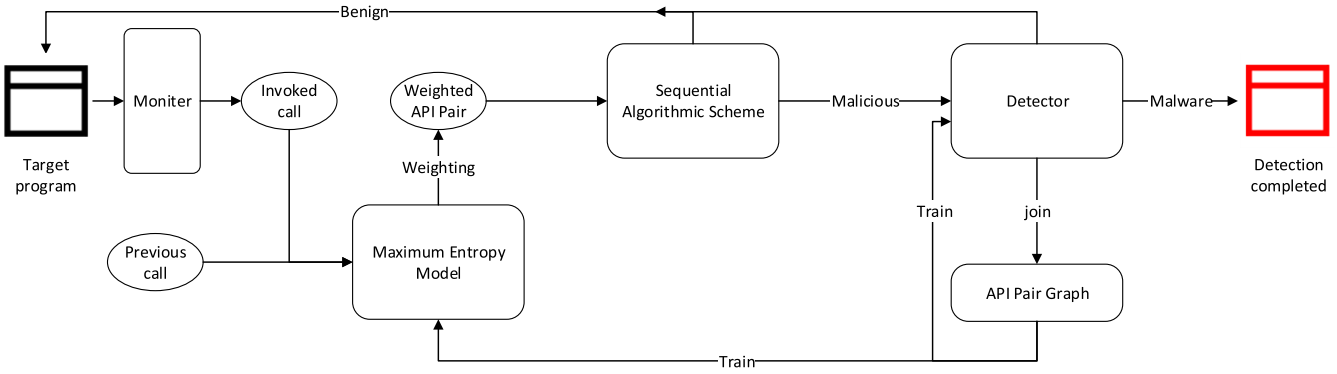


FIGURE 1. System design of the model.

IV. METHODOLOGY

Analysis of whole call graphs would be complex. It is too specific for a single malware and cannot extract general features for all malware detection. Meanwhile, learning from every single API in API call sequences would be lost characteristics of relevance among call sequences. Considering thoughts of Markov chains, the method would focus on analyzing connections between API in call sequences and reconstruct the call graph of malware.

In this section, our method of extract features, adaptive learning, optimization, and real-time detection would be proposed.

A. API PAIR GRAPHS

Common API-based malware detection used a complete system call sequence graph as inputs. However, the graph cannot apply to real-time detection, for the detection must identify every single behavior of samples. Another method concentrated on single API invoke as detection point, which will be regardless of important connection among API call sequences. In our model, an API pair would be adopted as meta for detection. The idea was based on the thought of Markov chain and has been evaluated in [11], [12], which defined each type of event as a system state, and the state was represented as a graph to describe the state of change [13]. When some events were executed in a specific order, it may clarify that some abnormal behaviors occurred.

Considering a sequence of invoked system-calls $\epsilon = \{c_1, c_2, c_3, c_4, \ldots, c_n\}$, where c_n referred to a system-call invoked. Then an API Pair Graphs could be constructed as $\rho = \{\langle s, c_1\rangle, \langle c_1, c_2\rangle, \ldots, \langle c_{n-1}, c_n\rangle, \langle c_n, e\rangle\}$, where s and e refers to a hypothetical call of start and end. The target is not focused on a single API or a full sequential call graph. Instead, the approach adopted API pairs as targets to represent behavior and the current state of a program.

Definition 1: An API Pair Graph $G = (S, E)$ refers to a directed graph, where S is a set of 2-length API call sequences. $E = \{e_{i,j} | c_i \xrightarrow{w_{i,j}} c_j, (c_i, c_j \in \epsilon)\}$, where $w_{i,j}$ means weight (sensitivity) of behavior $c_i \rightarrow c_j$.

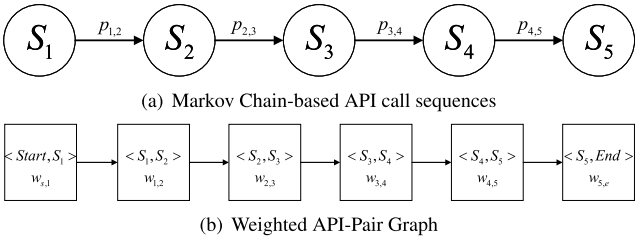


FIGURE 2. Structure of API call sequences and API-Pair Graph.

$E_{i,j}$ is a two-dimension set of API calls, so that an n-length call sequence could be transformed to a set 2-gram oriented graph. Figure. 2. showed the difference between HMM-based sequence and API Pair Graph, the transformation from the sequence to the graph could split complex chains into a group of two-dimension units, which could retain the connectivity of API calls.

Graph G was supposed to be Markov chains, which means the next behavior $E_{j,k}$ only associates with the current behavior $E_{i,j}$ and is unrelated with the previous part. However, in Markov chains, a weight $w_{i,j}$ of $c_i \rightarrow c_j$ refers to the probability from state s_i to s_j, and the probability could be computed as follows:

$$p_{i,j} = \frac{c_i \rightarrow c_j}{\sum c_i \rightarrow c_j} \tag{1}$$

The equation presented that the weights of Markov chains are only related to the frequency that edge $s_i \rightarrow s_j$ appeared in known datasets, so that the prediction ability of Markov chains is restricted by known samples. On the other hand, the identification of malware is more sophisticated, which could be influenced by many other factors. Moreover, malware updates very frequently to avoid the detection of the anti-malware solutions, which means past data would be outdated in just a few times, and simple probabilities of sequences would be a disadvantage. Besides, some crackers would add some confusing instructions to mislead detectors or even destroy the accuracy of the detectors; some related researches have had some results. All the conditions showed that Markov chains could only be considered as inputs; other approaches for filtering and detection are needed.

B. WEIGHT COMPUTING

Our approach of weighting is based on the Maximum Entropy model. The model is developed from entropy and maximum entropy principle. The concept of entropy in information science was proposed by Shannon [14], which could present uncertainty of a set of random variables. Assuming that X is a limited discrete random variable, and the probabilities of X could be described as $P(X = x_i) = p_i$ $(i = 1, 2, \ldots, n)$. Then an entropy H of X could be defined as follows:

$$H(X) = -\sum_{i=1}^{n} \log p_i \tag{2}$$

The maximum entropy principle is widely applied in probability model learning [15]. The thought of the principle is that the target model should satisfy known conditions, and all other parts would seem as equiprobable without additional information. The principle judged equiprobability with entropy maximization. Considering $H(X)$ in 2, $H(X)$ would meet the following expression:

$$0 \leq H(X) \leq \log |X|$$

The maximum entropy model is a classification model based on the maximum entropy principle. Giving a training dataset $D_T = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, the experience distribution of joint distribution (X, Y) and marginal distribution X are certain. Then the expectation of (X,Y) could be computed as follows:

$$E_{\tilde{P}}(f) = \sum_{x,y} \tilde{P}(x, y) f(x, y) \tag{3}$$

where $\tilde{P}(x, y)$ is the experience distribution of joint distribution (X, Y), $f(x, y)$ is a characterristic function of joint distribution (X, Y). Besides, the expectation of $P(Y|X)$ and $\tilde{X}$ would be:

$$E_P(f) = \sum_{x,y} \tilde{P}(x) P(Y|X) f(x, y) \tag{4}$$

If the model could get enough information from the dataset D_T, then $E_{\tilde{P}}(f)$ would be equal to $E_p(f)$, namely

$$\sum_{x,y} \tilde{P}(x, y) f(x, y) = \sum_{x,y} \tilde{P}(x) P(Y|X) f(x, y) \tag{5}$$

The equation (5) would be used as constraints of model learning. If the dataset has n characteristic functions $f_i(x, y)$, $i = 1, 2, \ldots, n$, then the model has n cocnstraints.

Definition 2: A maximum entropy model M is a model which have the highest conditional entropy $H(P)$ in a model set S, where

$$S \equiv \left\{ P \in P | E_{\tilde{P}}(f) = E_P(f) \right\} \tag{6}$$

and

$$H(P) = -\sum_{x,y} \tilde{P}(x) P(y|x) \log P(y|x) \tag{7}$$

The learning process of the maximum entropy model is a process of solving the maximum entropy model with (6) and (7). The learning could be formatted to constrained optimization, and Improved Iterative Scaling (IIS) is one of the most applied optimization algorithms for learning of maximum entropy model, and our algorithm is also based on the optimization algorithm. After the API chains transformed to API Pair Graph, the graph would be trained with weight computing algorithm and finally get a probable weight of being malicious and benign for each API pair.

Algorithm 1 Weight Computing Algorithm

Input: characteristic function $f_1, f2, \ldots, f_n$; experience distribution $\tilde{P}(X, Y)$; model $P_w(y|x)$.
Output: optimal weight w_{mi}, w_{bi}; optimal model P_{w^*}.

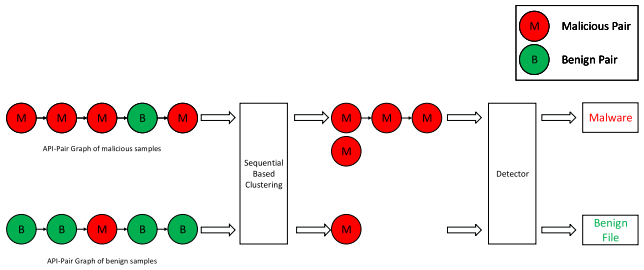
```
for each i ∈ {1, 2, ..., n} do
    w_mi = 0, w_bi = 0
end for
while w_*i is not converage do
    for each i ∈ {1, 2, ..., n} do
        f#(x, y) = Σ_{i=1}^{n} f_i(x, y)
        σ_i is the solution of the equation
        Σ_{x,y} P̃(x)P(y|x)f_i(x, y)exp(σ_i f#(x, y)) = E_P̃(f_i)
        w_mi = w_mi + σ_mi, w_bi = w_bi + σ_bi
    end for
end while
```

Weight Computing Algorithm: the proposed approach uses weighted API-Pair Graph to represent the relationship among API-Pair and API call sequences in each program. The model adopts the *Algorithm* 1 to compute and update the weight w_i of each pair p_i. The algorithm is a variant of the IIS model, where f_i presents the connection between API-Pair and properties of the program, malicious or benign. For each pair, the weight is the optimal solution of the formula that follows the maximum entropy principle, which corresponds with the occurrences of the pair in malicious and benign program [16], [17].

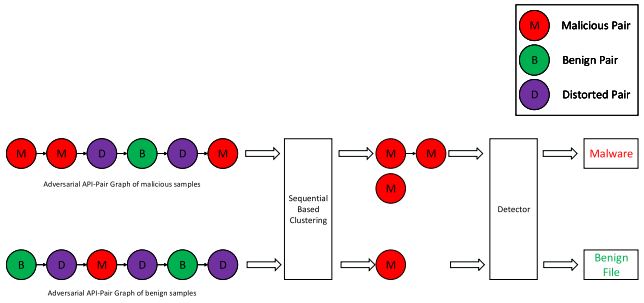
The maximum entropy model could predict the probability of a random event or event chain, which have been applied in some study [18], [19]. The result would meet all known constraints, and unknown conditions would be seemed as equiprobable events. In our study, the weight of each API pair can only be connected with known information, and it can update with an increase of datasets. Hence, the weight could adjust adaptively with the development tendency of malware.

C. CLUSTERING AND FILTERING

In this phase, we will present an adjusted Basic Sequential Algorithmic Scheme (BSAS) algorithm for weighted API-Pair, the proposed algorithm could be seen as a filter to screen those most relevant API pairs and remove those API calls which were added artificially to disguise detectors.



(a) Clustering of normal samples



(b) Clustering of adversarial samples

FIGURE 3. **Process of clustering graphs.**

The basic sequential algorithm scheme is a clustering algorithm that is based on the sequence of input data [20], which is widely used in the scheme of clustering [21] and data classification [20]. Generally, the constraints of the BSAS algorithm are the maximum number of clusters, the threshold of dissimilarity, and the dissimilarity that is mostly presented by the distance between units and clusters [22]. The thought of BSAS is to classify each unit to an existing cluster or create a new cluster for the unit [23]. The classification is related to the distance among clusters and units, and the distance would be decided by all units included.

Generally, malware in the same category usually have similar behavior patterns and execution sequences. Therefore, the appearance number of API invoked in the behavior chain are also related, and the API-Pair composed of those API would have adjacent positions and weights. With BSAS clusters, the API-Pair unrelated with malicious behaviors would be dropped and those in the same behavior chains would be retained, which would improve the efficiency of the model and reduce the interference and consumption.

The main principle of adversarial attack is to add noise to the samples so that it could hinder the identification of the model. The BSAS algorithm could filter the inputs have few relation with current API-Pair sequences. Hence, only API-pair correspond to malicious bahavior chains would be retained, and disguised items would be filtered. The diagram of the BSAS process is presented in Figure. 3.

In our method, the number of clusters would be set as two, which is malware and benign. The threshold is unneeded, and classification would be transformed into a comparison of the distance between new units and the central point of each cluster. After classification, the central point would be updated.

Algorithm 2 Clustering Algorithm

Input: API-Pair p_i in API-Pair Graph G;
Output: p_i if p_i in Malicious Cluster C_m else None.

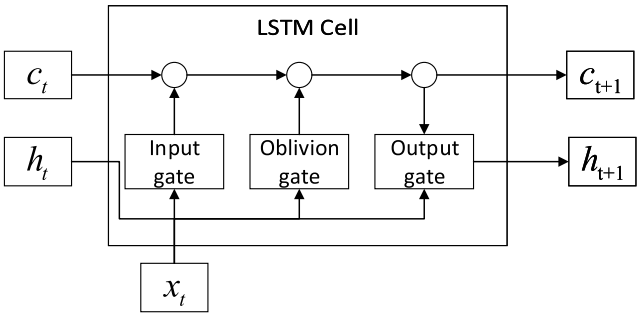
Initialize Malicious Cluster$\rightarrow C_m$, Benign Cluster$\rightarrow C_b$
for $p_i \in G, i \in (1, 2, \ldots, n)$ **do**
  if $C_m = \varnothing$ or $C_b = \varnothing$ **then**
    if $w_{mi} \leq w_{bi}$ **then**
      $C_m \leftarrow p_i$
    else
      $C_b \leftarrow p_i$
    end if
  else
    $center_m \leftarrow (\frac{\sum_{p_j} w_{mj}}{\sum_{i=1}^{n} p_j}, \frac{\sum_{p_j} w_{bj}}{\sum_{i=1}^{n} p_j}), p_j \in C_m$
    $center_b \leftarrow (\frac{\sum_{p_k} w_{mj}}{\sum_{i=1}^{n} p_k}, \frac{\sum_{p_k} w_{bk}}{\sum_{i=1}^{n} p_k}), p_k \in C_b$
    $d_{m,p_i} \leftarrow distance(center_m, p_i), d_{b,p_i} \leftarrow distance(center_b, p_i)$
    if $d_{m,p_i} \leq d_{b,p_i}$ **then**
      $C_m \leftarrow p_i$
    else
      $C_b \leftarrow p_i$
    end if
  end if
  if $p_i \in C_m$ **then**
    Return p_i
  end if
end for

Clustering Algorithm: The clustering algorithm would classify the pairs in the same API-Pair Graph with their sequences of appearance and the distance between their weights and the central point of clusters. Each pair would be clustered into the malicious cluster or the benign cluster, and the pair belonged to malicious cluster would be output for the next step of detection.

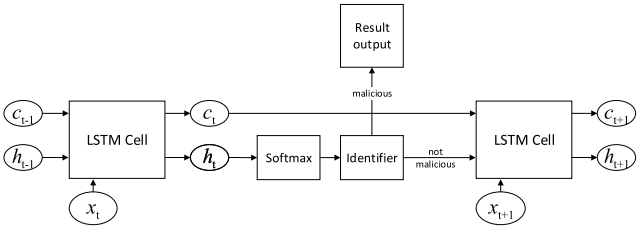
D. REAL-TIME DETECTION

The real-time detection approach of the model is based on Lone Short-Term Memory (LSTM) network. Severe researches have clarified that the model have a good effect in detecting sequence-based malicious data [24], [25] and real-time detection [26]. In our work, the structure of the network would be improved to adapt to the aim of instant identification. The input of the network would be API-Pair identified by the filter algorithm, and the sequential sequence of API-Pair could match with the execution of samples, so that the model could achieve the goal of real-time detection.

The common structure of LSTM is presented in Fig. 4-a. The structure is consists of a cell and three "gates" to process data. In the input phase, the sequence data x_t of time t will be pushed in and combined with the long memory data c_t and short memory data h_t to generate new data c_{t+1} and h_{t+1}. In the output phase, the current short memory data h_t



(a) Basic LSTM structure



(b) Improved LSTM structure

FIGURE 4. LSTM structure.

would be processed with full connection and softmax layer to generate the output matrix for identification.

Our improved LSTM structure is shown in Figure. 4-b. In order to achieve the goal of real-time detection, a function was added to determine whether the current sequence could be seen as malicious. If the function gives a positive result (malicious), the model will stop the process, and the output would be presented, else the model would suspend temporarily and wait for the next input.

V. EXPERIMENTS AND RESULTS

A. ENVIRONMENTS AND DATASETS

1) ENVIRONMENTS

The experiments are performed on a workstation with CPU of Intel Xeon E5-2603 and RAM of 32 G.B., running on Ubuntu 18.04 operation system. To record the call sequences of samples, Cuckoo Sandbox was adopted to monitor the behavior of the target program. A virtual machine with Windows 7 64-Bit was installed to run and monitor the program. Considering real-time monitor of API is complex, the call sequences were split to single API, and each API was involved one time to simulate real-time monitor and detection.

2) DATASET

In this work, we collected a dataset consists of 13624 samples, which have 6686 malware and 6938 benign samples, the concrete statistics were presented in Table. 1. The malicious samples came from VirusTotal [27] and VirusShare [28], and the benign samples are from system programs as well as the Internet. The benign dataset would be split into five parts evenly to fit malware dataset of each year, in our most experiments. The datasets would be set 80% for training and 20% for evaluation.

TABLE 1. Number of samples.

Dataset	Number
2016 Malware Dataset	1606
2017 Malware Dataset	1247
2018 Malware Dataset	1656
2019 Malware Dataset	888
2020 Malware Dataset	1289
Benign File Dataset	6938
Total	13624

TABLE 2. Top 20 API-Pairs.

API-Pair	Count
NtReadFile, NtReadFile	41188728
NtQuerySystemInformation, NtQuerySystemInformation	25934325
NtDelayExecution, NtDelayExecution	22073863
GetAsyncKeyState, GetAsyncKeyState	15279121
ReadProcessMemory, ReadProcessMemory	15047064
CryptHashData, CryptHashData	14011791
GetKeyState, GetKeyState	13113138
NtWriteFile, NtWriteFile	10556980
GetKeyState, GetAsyncKeyState	9239945
GetAsyncKeyState, GetKeyState	9239945
LdrGetProcedureAddress, LdrGetProcedureAddress	6413252
timeGetTime, timeGetTime	6340140
NtFreeVirtualMemory, NtFreeVirtualMemory	5209827
GetForegroundWindow, GetForegroundWindow	5127321
GetFileType, GetFileType	4910374
GetSystemMetrics, GetSystemMetrics	4835010
DeleteFileW, DeleteFileW	4632405
GetSystemTimeAsFileTime, GetSystemTimeAsFileTime	4614812
NtReadVirtualMemory, NtReadVirtualMemory	4425495
LdrLoadDll, SetErrorMode	4142028

B. EXPERIMENT STEP

1) GENERATING API-PAIR

Firstly, the invoked API call was transformed into a point. Then the point could combine with the last point and became a two-dimension point, which could be seen as API-Pair. The top ten appeared most API pair in experiments were presented in Table. 2.

2) TRAINING WEIGHT

In the weight training phase, each API-Pair Graph of samples was imported in the Maximum entropy model. The model would extract each pair and label of the graph and compute two weights, each weight connected with a pair and a label. Table. 3 presented ten API-pairs with the highest weight of being malicious and benign. The NaN in the table meant that the pair hadn't appeared in related samples. Some API pairs that have both malicious and benign weight are shown in Table. 4. The tables indicated that API could be quite distinct in the possibility of being malicious and benign.

In order to improve the accuracy of the identifier, the weight would be processed with normalization before detection. For each weight (w_{mi}, w_{bi}) of pair p_i, new weight(W_{mi}, W_{bi}) would be computed as:

$$W_{mi} = \frac{w_{mi}}{w_{mi} + w_{bi}} \tag{8}$$

$$W_{bi} = \frac{w_{bi}}{w_{mi} + w_{bi}} \tag{9}$$

TABLE 3. API-Pairs with highest weights.

Class	API-Pair	Malicious Weight	Benign Weight
Malicious pair	LdrLoadDll, NtResumeThread	6.93147180560073E-05	NaN
	NtCreateFile, CoInitializeSecurity	6.93147180559955E-05	NaN
	URLDownloadToFileW, NtCreateFile	6.93147180559955E-05	NaN
	NtProtectVirtualMemory, NtWriteFile	6.93147180559954E-05	NaN
	NtCreateFile, URLDownloadToFileW	6.93147180559951E-05	NaN
	NtOpenFile, NtDelayExecution	6.93147180559949E-05	NaN
	PRF, LdrLoadDll	6.93147180559948E-05	NaN
	LdrLoadDll, CoInitializeSecurity	6.93147180559947E-05	NaN
	NtCreateFile, OleInitialize	6.93147180559947E-05	NaN
	CIFrameElement_CreateElement, NtResumeThread	6.93147180559947E-05	NaN
	CreateProcessInternalW, NtWriteFile	6.93147180559947E-05	NaN
Benign pair	GetAsyncKeyState, GetAsyncKeyState	NaN	6.93147180559966E-05
	NtClose, NtQueryValueKey	NaN	6.93147180559955E-05
	RegOpenKeyExW, NtQueryValueKey	NaN	6.93147180559954E-05
	GetFileAttributesW, FindResourceExW	NaN	6.93147180559953E-05
	timeGetTime, GetForegroundWindow	NaN	6.93147180559953E-05
	NtEnumerateKey, NtClose	NaN	6.93147180559952E-05
	LdrGetDllHandle, FindResourceA	NaN	6.93147180559952E-05
	GetSystemMetrics, LoadStringW	NaN	6.93147180559952E-05
	NtClose, NtOpenDirectoryObject	NaN	6.93147180559952E-05
	OpenServiceA, OpenServiceA	NaN	6.93147180559952E-05
	NtCreateKey, NtQueryKey	NaN	6.93147180559951E-05

TABLE 4. API-Pairs with highest weights (two weights ver.).

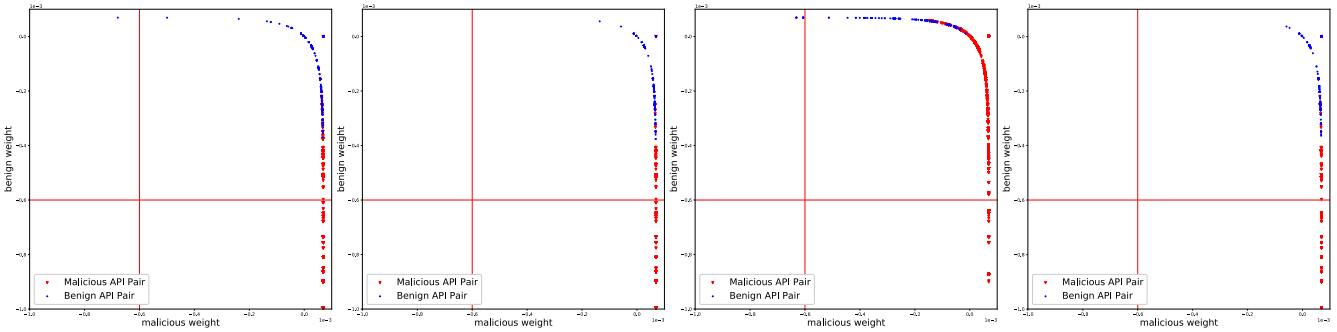
Class	API-Pair	Malicious Weight	Benign Weight
Malicious pair	NtCreateFile, NtOpenFile	6.93043887089331E-05	-8.48484083699043E-04
	NtCreateFile, CoInitializeEx	6.93004605172817E-05	-8.16256377301072E-04
	LdrLoadDll, NtOpenFile	6.92877851380088E-05	-7.52656369152654E-04
	NtCreateFile, CoUninitialize	6.92845566924681E-05	-7.41336733569528E-04
	NtOpenFile, LdrLoadDll	6.92741768300694E-05	-7.11766160351444E-04
	NtOpenFile, NtCreateFile	6.92665243769981E-05	-6.94479136598714E-04
	LdrLoadDll, CoUninitialize	6.92590335997946E-05	-6.80035564870549E-04
	NtOpenFile, NtAllocateVirtualMemory	6.92538028024739E-05	-6.71059923322954E-04
	NtProtectVirtualMemory, OleInitialize	6.92510845826390E-05	-6.66695679242923E-04
	NtProtectVirtualMemory, CoCreateInstance	6.92507180538076E-05	-6.66121514986156E-04
	NtCreateFile, CoCreateInstance	6.92504424399545E-05	-6.65691930765097E-04
Benign pair	GetCursorPos, GetCursorPos	-9.96690826165500E-04	6.93123716570861E-05
	RegEnumKeyExA, RegEnumKeyExA	-7.52940645783693E-04	6.92878616035785E-05
	timeGetTime, timeGetTime	-7.48134441198361E-04	6.92865391195753E-05
	NtProtectVirtualMemory, LdrGetProcedureAddress	-6.54678541076055E-04	6.92429563019261E-05
	NtEnumerateValueKey, NtClose	-6.48004456192667E-04	6.92380015198174E-05
	LoadResource, FindResourceW	-6.11368217983226E-04	6.92040373584726E-05
	GetSystemInfo, GetFileAttributesExW	-6.07534603108868E-04	6.91997094176709E-05
	NtOpenKeyEx, NtQueryValueKey	-5.93141593124331E-04	6.91818938624628E-05
	NtClose, GetCursorPos	-5.92958914338992E-04	6.91816508374217E-05
	NtQueryValueKey, NtQueryAttributesFile	-5.76832099579377E-04	6.91583458583770E-05
	RegEnumKeyExW, RegEnumKeyExW	-5.64803482991912E-04	6.91383407982208E-05

After normalization, most of weight would be situated in section $[-1, 1]$, which could improve the sensibility of the weight and decrease the computation consumption of the model.

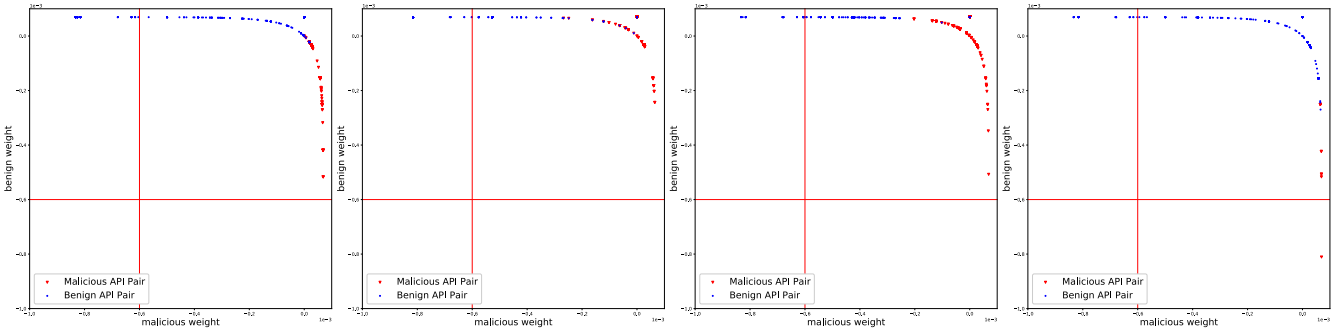
3) FILTERING

For every API-Pair P with weights wm and wb, we considered whole API-Pair Graph and split it into two clusters, C_m and C_b, where C_m referred to API pair that could be malicious or suspicious, and C_b included API Pari might be benign. Figure. 5 presented some classic samples clustered. In most cases, both malware and benign samples could invoke those suspicious API pairs. However, some regular patterns could be discovered between two kinds of samples. In a cluster map of benign samples, the weight of being benign rarely smaller than -2e6. While the weights in malware cluster maps always get the value or lower than it.

To show the effect of the Clustering algorithm intuitively, we collected 400 samples that have a call sequence length lower than 400. Then the samples were processed with BSAS



(a) Scatters of malicious files' clusters



(b) Scatters of benign files' clusters

FIGURE 5. Scatters of some target programs.

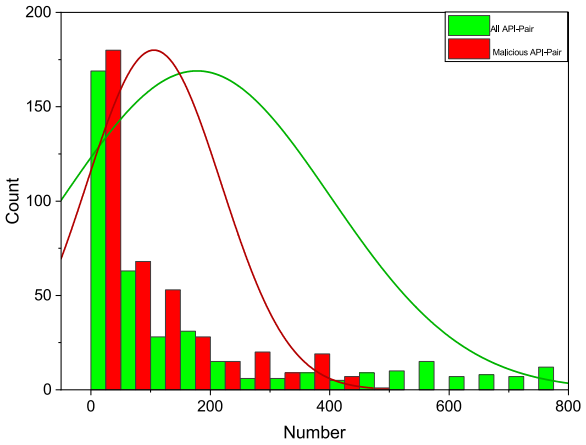


FIGURE 6. Statistics of API number.

clustering. The statistics of API-Pair and clustered malicious pair are presented in Figure. 5.

As shown in Figure. 6, the count of malicious API-Pair are higher than all API-Pair in the API-Pair number below 400. Besides, all number of malicious API-Pair were lower than 500. The results presented that BSAS clustering had a good effect in reducing the amount of data that needed to be detected.

4) DETECTION

In this section, some detection results obtained from our approach were presented and analyzed. The experiments included the following cases:

TABLE 5. Normal detection result.

Year	Accuracy	Precision	Recall	F1-Score
2016	0.9933	0.9969	0.9907	0.9938
2017	0.9792	0.9918	0.9640	0.9777
2018	0.9787	0.9819	0.9789	0.9804
2019	0.9715	0.9326	0.9940	0.9623
2020	0.8731	0.9167	0.8101	0.8601

1) normal detection for malware datasets, which consisted of malware from different years.
2) sequential detection for each dataset with time series, the aim of the experiment is to validate the adaptive learning of the model.
3) detection of unknown malware dataset to test the performance in predicting unknown samples.
4) detection for adversarial datasets to examine the robustness of the approach against adversarial evasion attacks.
5) performance of our model and other models to test the accuracy as well as time consumption.

C. RESULTS AND ANALYSIS

1) NORMAL DETECTION

In this phase, datasets of different years were trained and identified respectively to evaluate the general performance of our model. The results were depicted in Table. 5, where all datasets got an accuracy above 87%, and the accuracies except the 2020 year dataset were all above 97%. The result of the 2020 year dataset decreased obviously than other four

TABLE 6. Time series detection without adaptive learning.

Year	Accuracy	Precision	Recall	F1-Score
2016	0.9933	0.9969	0.9907	0.9938
2017	0.9716	0.9757	0.9640	0.9698
2018	0.9656	0.9507	0.9880	0.9690
2019	0.9364	0.9071	0.9336	0.9197
2020	0.7836	0.7536	0.8178	0.7844

TABLE 7. Time series detection with adaptive learning.

Year	Accuracy	Precision	Recall	F1-Score
2016	0.9933	0.9969	0.9907	0.9938
2017	0.9858	0.9911	0.9807	0.9859
2018	0.9879	0.9989	0.9778	0.9882
2019	0.9822	0.9962	0.9675	0.9816
2020	0.9713	0.9772	0.9640	0.9706

years' datasets, we supposed that some new obfuscation or disguised techniques might be applied between 2019 and 2020, which need to be verified further in the future.

2) TIME-SERIES DETECTION

In this phase, two experiments were adopted to evaluate the effect of adaptive learning in our models. Firstly, the 2016 year datasets were used as a primitive dataset to train the model, then each years' dataset as input for evaluation, after the evaluation and before the evaluation of the next year's dataset, each dataset would be learned by weighting model to update the weights automatically. As a comparison, the other experiment wouldn't learn from the dataset and update the weight adaptively, namely all datasets would only use weights learned from the dataset of 2016 year for detection.

Table. 6 presented the accuracy of every dataset without adaptive learning. The results showed a good rate on datasets of 2016 year, whereas the accuracy of other datasets decreased with the sequence of years, and the accuracy declined severely between the detection of 2019 and 2020 year, which might be attributed to the prevalence of new anti-detection techniques. The change of the accuracies showed that without the update of the model and features, the detection ability of a detector would decrease with the development of malware, which is in line to reality. Table. 7 showed the accuracy with the adaptive learning, where all evaluation kept an accuracy above 97%.

The comparison between separate detection and adaptive detection was showed in Figure. 7. The difference between detections increased with the pass of years, which could support the effectiveness of adaptive learning. Essentially, the result would also decrease with the development of time, while the accuracy fell much lower than the detection without the adaptive learning process. The findings could be thought that our dataset just simulated the consistent data with the samples collected from each year. However, the datasets are not consistent as well as abundant enough, so that some development of malware would not be presented in the datasets, and the model couldn't be learned, either. The thought is just one of many related hypotheses, which could be researched in the future.

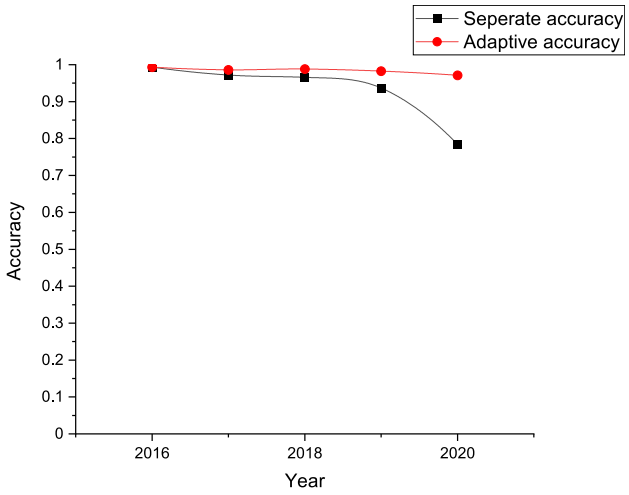


FIGURE 7. Accuracy comparison with datasets of different years.

3) UNKNOWN MALWARE DETECTION

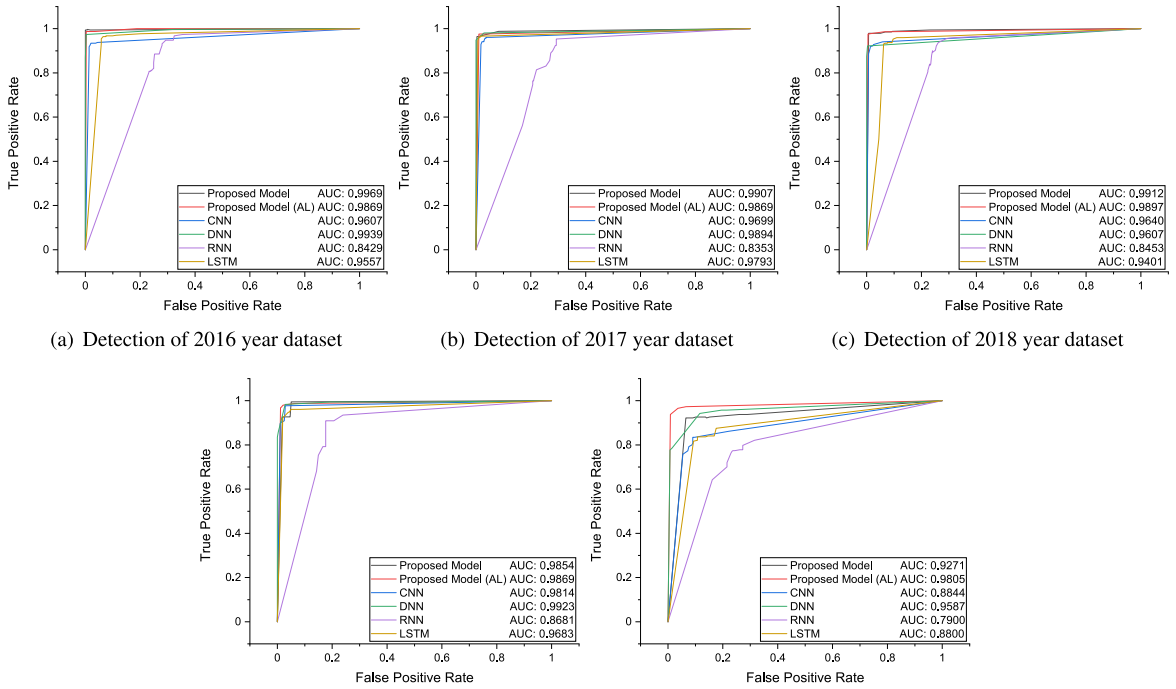
To test the performance of our model in detecting unknown malware, we collected 1664 samples from Ali Tianchi contest dataset to test the detection ability of the model on unknown malware. The results presented that our approach got an accuracy of 83.16% before adaptive training and 99.13% after the training. The experiment showed that the model has a detection capacity on unknown malware. However, only after enough samples were collected, and the model was trained adaptively so that the detection would have a good effect. The facts highlighted that the constant input of samples is necessary for our model so that it can improve the faculty of detection with the development of malware techniques.

4) PERFORMANCE EVALUATION

In order to test the performance of our method, an experiment was devised to compare the accuracy and time consumption between our approach and several deep learning models.

In this section, we adopted Convolutional Neural Network (CNN), Deep Neural Network (DNN), Recurrent Neural Network (RNN) and normal Long Short-Term Memory (LSTM) as a comparison to evaluate the performance of our approach. For the comparison approaches, The sources were API call sequences extracted from the execution of the programs. Then the sequences would be vectored and be processed with Primitive Components Analysis (PCA) model. The aim of the process is to reduce the dimension of the sequences and improve the performance of the detection. Finally, the processed data would be input to the comparison models for identification.

Table. 8 presented the detection results of each model. The results indicate that all model had a relatively high accuracy in detecting malware with API call sequences, and our approach got the highest accuracy in all detections. The Receiver Operating Characteristic (ROC) curves of each detection were shown in Figure. 8, it could be seen that our model had the best performance in 2016, 2017 and 2018 years' detection as



(a) Detection of 2016 year dataset (b) Detection of 2017 year dataset (c) Detection of 2018 year dataset

(d) Detection of 2019 year dataset (e) Detection of 2020 year dataset

FIGURE 8. ROC Curve of detection.

TABLE 8. Accuracy of detection for each model.

Year	Accuracy				
	Our model	CNN	DNN	RNN	LSTM
2016	0.9933	0.9449	0.9766	0.9805	0.9349
2017	0.9792	0.9677	0.9715	0.9756	0.9469
2018	0.9787	0.9491	0.9836	0.9805	0.9228
2019	0.9715	0.9538	0.9824	0.9707	0.9143
2020	0.8731	0.8675	0.9086	0.9122	0.8507

well as kept second-best performance in 2019 and 2020 years' detection.

Moreover, we evaluated the real-time detection for our method and other models. For technique reasons, the detections were implemented with simulation experiments. We collected execution time and invokes API numbers of our datasets to compute an average time of each call invoked, which is 0.051s. Then the datasets were processed with a specific program so that each API would be released after an average time. For comparison models of CNN, RNN and LSTM, since the models are based on static detection, the detection would activate after all API calls released.

The evaluation results were shown in Table. 9. For all datasets, the cost time of our approach could be less than 4.2% of other models in the best situation. The results validated the performance of our model and indicated the feasibility of real-time detection for malware.

5) DETECTION AGAINST ADVERSARIAL EVASION

As the conception of adversarial learning were proposed by In this subsection, a usual adversarial attack method of Fast Gradient Sign Method (FGSM) [29] was adopted to test our model's defending performance.

TABLE 9. Time consumption of detection for each model.

Year	Time (unit: second)				
	Our Model	CNN	DNN	RNN	LSTM
2016	2245.51	22352.38	22352.45	22352.96	22352.17
2017	1437.42	19934.33	19933.42	19933.47	19934.32
2018	964.70	22780.64	22782.01	22781.75	22780.83
2019	1320.04	16485.90	16485.25	16484.44	16484.35
2020	2386.98	20938.28	20937.89	20938.27	20936.61

TABLE 10. Accuracy of the models with FGSM adversarial samples.

Distortion	Accuracy				
	Proposed model	CNN	DNN	RNN	LSTM
0	0.9825	0.9713	0.9392	0.9005	0.9643
1	0.9716	0.9596	0.9332	0.7265	0.9519
2	0.9712	0.9452	0.9299	0.7094	0.9509
3	0.9700	0.9269	0.9222	0.6897	0.9516
4	0.9687	0.9085	0.9051	0.6490	0.9522
5	0.9691	0.8921	0.8918	0.5728	0.9526
6	0.9687	0.8794	0.8754	0.5097	0.9536
7	0.9691	0.8654	0.8584	0.4683	0.9529
8	0.9691	0.8520	0.8400	0.4539	0.9542
9	0.9691	0.8373	0.8206	0.4519	0.9536

In the test with FGSM, a distortion with upper limit $\epsilon \in [0, 9]$ was added to disguise the model. As a reference, the maximum and minimum value (weight) of data used in the proposed model are -18 and 280. Besides, the four models constructed in the previous section were also evaluated for comparison, while the maximum and minimum value (processed with PCA) of data are -6040 and 13880.

The result of the test with FGSM was shown in Table. 10. From the table, it can be seen that the accuracy of the proposed model decreased from 98.99% to 96.91% with the growth of ϵ, which is relatively stable. On the contrast, the accuracy of CNN, DNN and RNN decrease severely. What

is interesting about the result is that the accuracy of LSTM decreased slower than our model, which was finally stabilized near 95%. The reason for the phenomenon could be that the adopted LSTM model has a multi-layer structure, which has been verified by related works that could improve the robustness of deep learning model against adversarial attack. However, another similar model of RNN decreased more severely among models, which might be related to the gradients characteristics which needed to be leaned further in the future.

The result of the experiments clarified that our model has a certain effect in defending usual adversarial attacks, which could retain an accuracy above 96.91%.

VI. DISCUSSION

A. COMPARISON WITH EXISTING APPROACHES

In this subsection, we compared other API-based methods with our model in identifying malware, tough the works used different datasets which were mostly collected by themselves. Some of their works presented results with TPR and FPR, others provided accuracy, precision, recall and F1-Score. Table. 11 presented a comparison of different evaluations. As shown in Table. 11, the proposed approach showed approximate results with [31] in TPR and FPR. Furthermore, our approach outperformed other works in terms of accuracy, precision and F1-Score. The reason for our model had a better performance in these indexes is the weight of our data would be updated adaptively with the increase of malware datasets. Also, we adopted a sequential cluster algorithm to filter the data, so that our model could avoid interference from unrelated features, which would consequently improve the specificity and identification rate of the detection.

Due to the clustering process, our model might be more sensitive with several specific features and recognized some benign targets as malware, such as critical system programs. Hence, our model had a relatively low performance in FPR and Recall than [31]. However, [31] adopted features based on API call sequences after execution, indicating it must wait for the run of programs finished. The distinction between malware and systems is respectively more complex. Our method could achieve the goal of real-time detection in principle, which could realize both reducing detecting time and prevent malicious targets from causing more damages when it was implemented in practical application.

B. LIMITATION

In this section, some shortages of the model would be discussed.

Firstly, the proposed model needs the weight of the API Pair to detect the program. So that the weighting approach, the maximum entropy model must keep a balance between the training of malware and benign samples. In this article, to keep the approximate number of malware and benign datasets, the number of samples in our experiments have to be constricted to a small number, if the balance between malware and benign was broken out, the model would lose its use. Fortunately, in the production environment, the detected malware and benign programs are tremendous enough to avoid the problem.

TABLE 11. Comparison with proposed model and other existing approaches.

Detection Approach	TPR	FPR	Accuracy	Precision	Recall	F1-Score
Smita et.al. [30]	0.9430	0.0460	0.9542	None	None	None
Tang et.al. [31]	0.9930	0.0009	None	0.9934	0.9930	0.9932
Zhang et al. [32]	None	None	0.9510	0.9570	0.9430	0.9500
Han et al. [33]	None	None	0.9439	0.9368	0.9311	0.9338
Burnap et al. [34]	None	None	0.9376	0.9460	0.9300	0.9380
Xiao et al. [35]	None	None	None	0.9860	0.9920	0.9890
Zhang et al. [32]	None	0.0410	0.9510	0.9570	0.9430	0.9500
Proposed approach	0.9907	0.0036	0.9933	0.9969	0.9907	0.9938

Besides, in the filtering phase, some benign samples were identified having too many malicious API pairs, while some malware were just detected few malicious API pairs. The reason for the phenomenon caused might because many benign programs also invoked sensitive API chains, especially many system programs, which is a considerable part of our datasets. Also, some malware just call few critical API to reach their goals, such as some shells or console trojans. Limited by collection abilities for benign samples, some further work should be implemented to check our assumptions.

Moreover, the model requires relatively high monitoring demands, such as malware must be compliant with target computer circumstances, and the monitor must work well. Without the requirements, some anomaly conditions, like malware invoked specific API endlessly or the monitor lost some invoked calls, would interfere with the result of the detection. In order to overcome these shortages, a fuzzy detector or similarity analyzer could be designed to enhance the robustness of the model. However, the extra phases would cost additional computation time and might decrease the accuracy of the identification.

VII. RELATED WORKS

A. API BASED MALWARE DETECTION

In recent years, many researchers focused on malware detection based on API sequences or graphs, especially connected with machine learning. Some works have been presented as follows:

Naval *et al.* [30] proposed a new model based on improved API sequences. They proposed a definition called Ordered System-Call Graph (OSCG) for information mining of API sequences. Besides, they applied asymptotic equipartition property (AEP) on the API call sequences to extract those more significant series. The approach was tested by a series of experiments, and the result was compared with existing malware detectors. The performance presented in the solution is effective in detecting malware.

Tang and Qian [31] proposed a new approach of static malware detection based on API call sequences. They firstly extract API sequences with dynamic analysis, then color mapping was used to transform the sequences to feature images that could represent malware behavior. Finally, a Convolutional neural network (CNN) was adopted to classify the

malware into nine malware families. The result showed the efficiency and classification statics all get over 99% with the FPR. is under 0.1%.

Xiaofeng *et al.* [36] proposed two approaches for malware detection based on API sequences. The first part is to extract features from API sequences and train a model based on random forest for classification. In the second part, they adopted an improved LSTM model to learn the semantic of API calls and detect malicious targets. The results of experiments showed that both approaches worked well, and the framework combined two models had a better performance, which had an accuracy of 96.7% in classification. Ma *et al.* [37] analyzed malicious malware and designed a detection framework based on API fragments. The framework utilized sliding window operation to split API sequences into an N-length API fragment. Then the fragments were learned by an LSTM-based detection model. Finally, the detection result would be filtered with a threshold. The author also researched the relationship between lengths and threshold. The result presented that the framework is effective when the length and threshold are appropriate, and the accuracy of detection could get a maximum of 97.34%.

Wang *et al.* [38] proposed a malware detection framework named ''LSCDroid'', the framework could learn features from datasets automatically as well as classify malware into different families based on feature sets. In this article, they firstly generated an API sequence graph. Then the graphs were used to compute the weight of each API invocation based on entropy. After that, the weighted graphs were adopted for similarity calculation, and the feature set was extracted. Finally, the feature set was learned by a machine learning-based model. The result proved the effectiveness of the framework, which had an accuracy of 98% for detection and 96% for classification.

B. REAL-TIME MALWARE DETECTION

Over the past few decades, most research in real-time detection has emphasized on analytics of traffics. However, real-time detection for malware is relatively few, and most of the researches about real-time malware detection also assisted with static detection. Some of the related particles are described as follows.

Chen *et al.* [39] focused on the real-time detection for Twitter spam. They primarily extract statistical features from spam mail and normal mail. Then a scheme was adopted, which consists of a learning layer and a training layer. The learning layer could find changes in unlabeled spam tweets and transfer the changed features to the training layer. The training layers could learn new features in the training step. The results showed that the scheme could increase the detection rate of spam in real-world detection. The research concentrated on spam. However, the thought of the study had implications for real-time malware detection.

Kim *et al.* [40] have studied on real-time detection of ransomware and protection method. The approach analyzed the operation procedure of the system and applied access control on significant operations. Besides, a whitelist was utilized to manage the program getting access to the operations. The method had a good performance in detecting and preventing ransomware. However, the consumption is relatively high, which could have a maximum CPU usage of over 70% and memory of usage over 2500 MB. In conclusion, the approach could only available in the host, which seen security as the highest priority, and it is not reliable in common production environments.

Gharib and Ghorbani [41] proposed a framework named ''DNA-Droid'' to detect attacks from ransomware. The framework consisted of two layers. One is a static layer to extract text and image information from samples for classification and detection. The other is a dynamic layer that would run programs in a sandbox to get API sequences and transform the sequences to DNAs, a specific signature for identifying a family of malware. The approach had a good precision in detecting malware with a rate of over 98%, which performed better than existing approaches.

C. ADVERSARIAL ATTACK AGAINST MACHINE LEARNING

Much of the current literature on security pays particular attention to adversarial attacks to machine learning-based malware detection. Since machine learning, especially deep learning technique has been applied widely in most of the fields, the risk and potential attacks has become a hot spot for research. Chen *et al.* proposed a novel approach that could attack the detection model by generating adversarial samples. The approach applied optional perturbations onto APK files and successfully deceived target machine learning detectors. The authors used some current detectors to identify the model, and the result presented effectiveness in disguising the detection models, which decreased the accuracy of MaMaDroid from 96% to 0%, and Derbin from 97% to 0%. Besides, the authors advised two approaches to defend adversarial attacks and tested with experiments. The conclusion is that both adversarial learning and ensemble learning methods have effects on protecting machine learning-based models from adversarial evasions.

Stokes *et al.* [42] investigated several adversarial attacks and preserving methods, then an approach of the weight decay defence was proposed and evaluated. The experiments presented that all defending approaches could protect the models from decreasing accuracy significantly when confronted with adversarial evasion attacks. Besides, the authors also discovered although adding hidden layers could not improve the accuracy of detection evidently, it could increase the robustness of the model to adversarial attacks.

Demetrio *et al.* [43] offered a state-of-art attack towards black-box detectors. The approach utilized several manipulations of protecting semantics to solve validation steps based on computation. The authors also transformed attack into a constrained minimization problem. The method had been tested by some static detectors and have a good performance. Besides, the method was also examined by several commercial antivirus solutions, and the results showed some of them

could be evaded by the attack. Finally, the limitation of the attack was discussed that it might not work on the detection method based on dynamic analysis, which might be studied in the future.

Chen *et al.* [44] concentrated on the method of adversarial attacking and proposed a new framework named ''KuafuDet'' to solve the problem. The paper firstly studied the viability of adversarial attacks against machine learning detection models with constructed misleading malware samples. The authors introduced several prevailing methods of generating adversarial samples and concluded current challenges to defend adversarial attacks. Then the paper proposed KuafuDet, a model that has strong robustness against adversarial attacks. The model consisted of two phases: the primary phase utilized a similarity-based approach to filter suspicious samples and got very benign and malicious individuals for the next training. Then a detector was adopted, which was composed of an online classification model and offline training model. The offline model would train data sets to obtain features of the application, and the feature would be input into the online classifier for classification. The experimental results showed that the model had a good performance in suffering attacks against the model, which reduced false negatives as well as improve the accuracy of detection by at least 15%. The effectiveness of the model has been proved by experiments, and all three tested approaches had stable results in detection.

Chen *et al.* [45] studied Android malware detectors based on machine learning and API sequences. They concluded several possible attacks that the models could suffer from attackers. Besides, the authors proposed an approach of attacks based on API call relevance of malware and benign applications. The test showed that the approach could be applicable in different scenarios. Finally, an adversarial model was designed to defend evasion attacks. The model used an adversarial learning method to improve robustness against evasion attacks. Also, the model maximized the cost of evasion and minimized the loss of classification. The comprehensive experiments were based on real sample datasets, and the results presented that the model was robust enough against the evasion attacks.

VIII. CONCLUSION

In this article, we proposed an adaptive approach for real-time detecting malware using API call sequences invoked by the target program. An API-Pair Graph was constructed so that a sequence could be transformed into a graph with the smallest unit of behavior chains; besides, a Maximum Entropy Model was implemented to compute the weight of each pair to be malicious or benign, which could keep the max information for uncertainty, and learn adaptively from existed data. We used an optimized cluster algorithm to increase the accuracy of the detection and save time, and the final data would be detected by an improved LSTM structure, which could give results instantly after an API was input. Finally, all the data would be transited to the weighting model to update the weight data so that the model could learn adaptively with the development of malware. We collected four datasets from separate years to test the approach, and some simulated datasets were used to examine the performance of the model towards unknown malware. Results showed that our proposed approach could reach the goal of real-time detection theoretically while keeping an ideal accuracy. Also, our model could keep high rates with development of malware and have an effort in identifying unknown malware. Additionally, our approach could keep resistance against a common adversarial attack, which could prove the robustness of our model.

Our work also has some disadvantages, which need to be revised in our next work. Firstly, limited with time and energy, the characteristic of real-time detection can only be evaluated by simulating experiment, though the result is ideal. Besides, our method has a restrict requirement in input data, which must meet an approximate number of malicious and benign datasets, which could be improved with several advancements.

In the future, we will focus on the practical implementation of the model to achieve the goal of application in industry. Besides, some related solutions would be reproduced and examined with the same dataset in our work to compare the performance more precisely. Furthermore, we will learn further in defence of machine-learning-based malware detection against adversarial attacks.

ACKNOWLEDGMENT

The authors would be very grateful for getting the Ali TianChi Malware dataset from Aliyun computing team. Besides, they would like to thank VirusTotal and VirusShare for granting them complimentary access to malicious samples. They would also like to thank the High-Performance Computing Center, Lanzhou University, Lanzhou, China, for providing the computation supports to the experiments.

REFERENCES

[1] O. Aslan and R. Samet, ''A comprehensive review on malware detection approaches,'' *IEEE Access*, vol. 8, pp. 6249–6271, 2020.

[2] R. Tahir and V. U. o. P. Department of Computer Science, ''A study on malware and malware detection techniques,'' *Int. J. Edu. Manage. Eng.*, vol. 8, no. 2, pp. 20–30, Mar. 2018.

[3] I. You and K. Yim, ''Malware obfuscation techniques: A brief survey,'' in *Proc. Int. Conf. Broadband, Wireless Comput., Commun. Appl.*, 2010, pp. 297–300.

[4] A. A. Selcuk, F. Orhan, and B. Batur, ''Undecidable problems in malware analysis,'' in *Proc. 12th Int. Conf. Internet Technol. Secured Trans. (ICITST)*, Dec. 2017, pp. 494–497.

[5] O. P. Samantray, S. N. Tripathy, and S. K. Das, ''A study to understand malware behavior through malware analysis,'' in *Proc. IEEE Int. Conf. Syst., Comput., Autom. Netw. (ICSCAN)*, Mar. 2019, pp. 1–5.

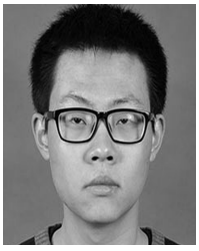
[6] *Internet Security Threat Report Volume 24*. Symantec. Accessed: 2019. [Online]. Available: https://docs.broadcom.com/doc/istr-24-2019-en

[7] *Mcafee Labs Threats Reports August 2019*. McAfee Labs. Accessed: Aug. 2019. [Online]. Available: https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-aug-2019.pdf

[8] S. Kok, A. Abdullah, N. Jhanjhi, and M. Supramaniam, ''Ransomware, threat and detection techniques: A review,'' *Int. J. Comput. Sci. Netw. Secur.*, vol. 19, no. 2, p. 136, 2019.

[9] Z. Bazrafshan, H. Hashemi, S. M. H. Fard, and A. Hamzeh, ''A survey on heuristic malware detection techniques,'' in *Proc. 5th Conf. Inf. Knowl. Technol.*, May 2013, pp. 113–120.

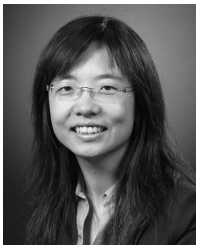
[10] S. Alqurashi and O. Batarfi, "A comparison between API call sequences and opcode sequences as reflectors of malware behavior," in *Proc. 12th Int. Conf. for Internet Technol. Secured Trans. (ICITST)*, Dec. 2017, pp. 105–110.
[11] F. Al Shamsi, W. L. Woon, and Z. Aung, "Discovering similarities in malware behaviors by clustering of API call sequences," in *Proc. Int. Conf. Neural Inf. Process.* Berlin, Germany: Springer-Verlag, 2018, pp. 122–133.
[12] N. Huang, M. Xu, N. Zheng, T. Qiao, and K.-K.-R. Choo, "Deep Android malware classification with API-based feature graph," in *Proc. 18th IEEE Int. Conf. Trust, Secur. Privacy Comput. Commun., 13th IEEE Int. Conf. Big Data Sci. Eng. (TrustCom/BigDataSE)*, Aug. 2019, pp. 296–303.
[13] D. Chen, "Anomaly detection boundary based on the moving averages of Markov chain model," in *Proc. 12th Int. Conf. Fuzzy Syst. Knowl. Discovery (FSKD)*, Aug. 2015, pp. 1532–1536.
[14] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, no. 3, pp. 379–423, Jul./Oct. 1948.
[15] X. Zhang, Q. Duan, and H. Yang, "Scale detection based on maximum entropy principle," in *Proc. 24th Int. Conf. Autom. Comput. (ICAC)*, Sep. 2018, pp. 1–6.
[16] W. Yaqi, X. Yonghai, and W. Jinhao, "The maximum entropy probability distribution and the improvement of its solving method for negative sequence current of a traction substation," in *Proc. China Int. Conf. Electr. Distrib. (CICED)*, Aug. 2016, pp. 1–5.
[17] B. Luo, W. Wang, Y. Jia, and W. Gao, "A segmentation method for spotted-partten damaged Thangka image combining grayscale morphology with maximum entropy threshold," in *Proc. 6th Int. Congr. Image Signal Process. (CISP)*, vol. 1, Dec. 2013, pp. 561–565.
[18] H. J. Landau, "Maximum entropy and maximum likelihood in spectral estimation," *IEEE Trans. Inf. Theory*, vol. 44, no. 3, pp. 1332–1336, May 1998.
[19] C. Yin, J. Xi, and J. Wang, "The research of text classification technology based on improved maximum entropy model," in *Proc. 1st Int. Conf. Comput. Intell. Theory, Syst. Appl.*, 2015, pp. 142–145.
[20] N. Ahmadi and R. Berangi, "A basic sequential algorithmic scheme approach for classification of modulation based on neural network," in *Proc. Int. Conf. Comput. Commun. Eng.*, May 2008, pp. 565–569.
[21] X. Li, X.-C. Cong, G.-J. Wen, Y. Yang, Q. Zhang, and Q. Wan, "Anisotropic image formation based on basic sequential algorithmic scheme and non-quadratic regularization," in *Proc. IEEE Int. Conf. Commun. Problem-Solving (ICCP)*, Oct. 2015, pp. 488–491.
[22] M. Parthiban and P. Damodharan, "Basics sequential algorithmic scheme clustering by exploring inter and intra task co-relations," *Int. J. Eng. Technol. Sci.*, vol. 2, no. 11, Nov. 2015.
[23] W. M. Ashour, R. Z. Muqat, A. B. AlQazzaz, and S. R. AbdElnabi, "Improve basic sequential algorithm scheme using ant colony algorithm," in *Proc. IEEE 7th Palestinian Int. Conf. Electr. Comput. Eng. (PICECE)*, Mar. 2019, pp. 1–6.
[24] T. Mu, H. Chen, J. Du, and A. Xu, "An Android malware detection method using deep learning based on API calls," in *Proc. IEEE 3rd Adv. Inf. Manage., Communicates, Electron. Autom. Control Conf. (IMCEC)*, Oct. 2019, pp. 2001–2004.
[25] Y. Fang, C. Huang, L. Liu, and M. Xue, "Research on malicious javascript detection technology based on LSTM," *IEEE Access*, vol. 6, pp. 59118–59125, 2018.
[26] J. Chatrath, P. Gupta, P. Ahuja, A. Goel, and S. M. Arora, "Real time human face detection and tracking," in *Proc. Int. Conf. Signal Process. Integr. Netw. (SPIN)*, Feb. 2014, pp. 705–710.
[27] V. Total. *Virustotal-Free Online Virus, Malware and URL Scanner*. Accessed: 2020. [Online]. Available: http:/www.virustotal.com
[28] VirusShare. *Virusshare. Com–Because Sharing is Caring*. Accessed: 2020. [Online]. Available: http://www.virusshare.com
[29] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," 2014, *arXiv:1412.6572*. [Online]. Available: http://arxiv.org/abs/1412.6572
[30] S. Naval, V. Laxmi, M. Rajarajan, M. S. Gaur, and M. Conti, "Employing program semantics for malware detection," *IEEE Trans. Inf. Forensics Security*, vol. 10, no. 12, pp. 2591–2604, Dec. 2015.
[31] M. Tang and Q. Qian, "Dynamic API call sequence visualisation for malware classification," *IET Inf. Secur.*, vol. 13, no. 4, pp. 367–377, Jul. 2019.
[32] J. Zhang, Z. Qin, H. Yin, L. Ou, and K. Zhang, "A feature-hybrid malware variants detection using CNN based opcode embedding and BPNN based API embedding," *Comput. Secur.*, vol. 84, pp. 376–392, Jul. 2019. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0167404818312902
[33] W. Han, J. Xue, Y. Wang, L. Huang, Z. Kong, and L. Mao, "MalDAE: Detecting and explaining malware based on correlation and fusion of static and dynamic characteristics," *Comput. Secur.*, vol. 83, pp. 208–233, Jun. 2019. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S016740481831246X
[34] P. Burnap, R. French, F. Turner, and K. Jones, "Malware classification using self organising feature maps and machine activity data," *Comput. Secur.*, vol. 73, pp. 399–410, Mar. 2018. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0167404817302535
[35] F. Xiao, Z. Lin, Y. Sun, and Y. Ma, "Malware detection based on deep learning of behavior graphs," *Math. Problems Eng.*, vol. 2019, pp. 1–10, Feb. 2019. [Online]. Available: https://www.hindawi.com/journals/mpe/2019/8195395/
[36] L. Xiaofeng, J. Fangshuo, Z. Xiao, Y. Shengwei, S. Jing, and P. Lio, "ASSCA: API sequence and statistics features combined architecture for malware detection," *Comput. Netw.*, vol. 157, pp. 99–111, Jul. 2019.
[37] X. Ma, S. Guo, W. Bai, J. Chen, S. Xia, and Z. Pan, "An API semantics-aware malware detection method based on deep learning," *Secur. Commun. Netw.*, vol. 2019, pp. 1–9, Nov. 2019.
[38] W. Wang, J. Wei, S. Zhang, and X. Luo, "LSCDroid: Malware detection based on local sensitive API invocation sequences," *IEEE Trans. Rel.*, vol. 69, no. 1, pp. 174–187, Mar. 2020.
[39] C. Chen, Y. Wang, J. Zhang, Y. Xiang, W. Zhou, and G. Min, "Statistical features-based real-time detection of drifted Twitter spam," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 4, pp. 914–925, Apr. 2017.
[40] D.-Y. Kim, G.-Y. Choi, and J.-H. Lee, "White list-based ransomware real-time detection and prevention for user device protection," in *Proc. IEEE Int. Conf. Consum. Electron. (ICCE)*, Jan. 2018, pp. 1–5.
[41] A. Gharib and A. Ghorbani, "Dna-droid: A real-time Android ransomware detection framework," in *Proc. Int. Conf. Netw. Syst. Secur.* Berlin, Germany: Springer-Verlag, 2017, pp. 184–198.
[42] J. W. Stokes, D. Wang, M. Marinescu, M. Marino, and B. Bussone, "Attack and defense of dynamic analysis-based, adversarial neural malware classification models," 2017, *arXiv:1712.05919*. [Online]. Available: http://arxiv.org/abs/1712.05919
[43] L. Demetrio, B. Biggio, G. Lagorio, F. Roli, and A. Armando, "Functionality-preserving black-box optimization of adversarial windows malware," 2020, *arXiv:2003.13526*. [Online]. Available: http://arxiv.org/abs/2003.13526
[44] S. Chen, M. Xue, L. Fan, S. Hao, L. Xu, H. Zhu, and B. Li, "Automated poisoning attacks and defenses in malware detection systems: An adversarial machine learning approach," *Comput. Secur.*, vol. 73, pp. 326–344, Mar. 2018.
[45] L. Chen, S. Hou, Y. Ye, and L. Chen, *An Adversarial Machine Learning Model Against Android Malware Evasion Attacks* (Lecture Notes in Computer Science). Cham, Switzerland: Springer, 2017, ch. 5, pp. 43–55.



SHAOJIE YANG received the bachelor's degree in geographic information science from Jilin University, in 2018. He is currently pursuing the master's degree with the Next Generation Internet Laboratory, Lanzhou University. His research interests include cyber security, reverse engineering, and machine learning.

SHANXI LI (Member, IEEE) received the B.S. and M.S. degrees in computer sciences from Lanzhou University, in 2002 and 2009, respectively, where he is currently pursuing the Ph.D. degree with the School of Information Science and Engineering. His research interests include computer networks, computer security, artificial intelligence, and machine learning.

WENBO CHEN received the B.S. and M.S. degrees in mechanics and the Ph.D. degree in applied mathematics from Lanzhou University, in 1993, 1996, and 2012, respectively. He is currently an Associate Professor with Lanzhou University, where he is also working with the School of Information Science and Engineering. His research interests include next-generation Internet, high-performance computing, artificial intelligence, and machine learning.



YUHONG LIU (Member, IEEE) received the B.S. and M.S. degrees from the Beijing University of Posts and Telecommunications, in 2004 and 2007, respectively, and the Ph.D. degree from the University of Rhode Island, in 2012. She is currently an Assistant Professor with the Department of Computer Engineering, Santa Clara University. With expertise in trustworthy computing and cyber security, her research interests include developing trust models and applying them on emerging applications, such as online social media, cyber-physical systems, and cloud computing. She was a recipient of the 2013 University of Rhode Island Graduate School Excellence in Doctoral Research Award and the Best Paper Award at the 9th International Conference on Ubi-Media Computing (UMEDIA 2016). Her work on securing online reputation systems received the Best Paper Award at the IEEE International Conference on Social Computing 2010 (acceptance rate = 13%).