Hindawi
Mathematical Problems in Engineering
Volume 2019, Article ID 8195395, 10 pages
https://doi.org/10.1155/2019/8195395

Research Article

Malware Detection Based on Deep Learning of Behavior Graphs

Fei Xiao,[1,2,3] **Zhaowen Lin**,[1,2,3] **Yi Sun**,[2,3,4] **and Yan Ma**[1]

[1] *Network and Information Center, Institute of Network Technology, Beijing University of Posts and Telecommunications, Beijing, 100876, China*
[2] *Science and Technology on Information Transmission and Dissemination in Communication Networks Laboratory, Shijiazhuang, 050081, China*
[3] *National Engineering Laboratory for Mobile Network Security, Beijing University of Posts and Telecommunications, Beijing, 100876, China*
[4] *Network and Information Center, Institute of Network Technology/Institute of Sensing Technology and Business, Beijing University of Posts and Communications, Beijing, 100000, China*

Correspondence should be addressed to Yi Sun; sybupt@bupt.edu.cn

Received 26 October 2018; Revised 15 January 2019; Accepted 21 January 2019; Published 11 February 2019

Academic Editor: Luis Martínez

The Internet of Things (IoT) provides various benefits, which makes smart device even closer. With more and more smart devices in IoT, security is not a one-device affair. Many attacks targeted at traditional computers in IoT environment may also aim at other IoT devices. In this paper, we consider an approach to protect IoT devices from being attacked by local computers. In response to this issue, we propose a novel behavior-based deep learning framework (BDLF) which is built in cloud platform for detecting malware in IoT environment. In the proposed BDLF, we first construct behavior graphs to provide efficient information of malware behaviors using extracted API calls. We then use a neural network-Stacked AutoEncoders (SAEs) for extracting high-level features from behavior graphs. The layers of SAEs are inserted one after another and the last layer is connected to some added classifiers. The architecture of the SAEs is 6,000-2,000-500. The experiment results demonstrate that the proposed BDLF can learn the semantics of higher-level malicious behaviors from behavior graphs and further increase the average detection precision by 1.5%.

1. Introduction

A large number of malware variants have been automatically generated per day. Recent Symantec report [1] shows that new pieces of malware grew by 36 percent from the year before in 2015 with total samples exceeding 430 million. Exponential growth of malware caused a considerable threat in our daily life.

Traditional computers bring a lot of attacks in IoT environment. Malware attacks computers and uses the infected computers to attack other connected devices in IoT environment. For example, Trojan.Mirai.1 which is the variant of Mirai can infect windows hosts and utilize these hosts to infect other devices. The infected windows can steal confidential information and transform the influenced devices into a botnet to launch a new Distributed Denial of Service (DDoS) attack. Many current traditional computers' malware attacks may also extend to other IoT devices. Unfortunately, there are no ideal solutions to avoid Mirai and other IoT threats. One approach aims to weaken these threats by protecting the security of traditional computers in IoT environment.

The fast-growing samples bring a large number of demands for malware detection in IoT environment [2–4]. With so many sophisticated malware samples, plenty of researches have been concentrated on proposing miscellaneous malware detection methods to mitigate the rapid growth of malware. Malware detection can be divided into two main methods: static malware detection and dynamic malware detection [5, 6]. Static malware detection also refers to signature-based malware detection which examines the content of malicious binary without actually executing malware samples. Signature-based malware detection is able to obtain full execution path. However, it can be easily evaded by obfuscation techniques. In addition, signature-based malware detection requires prior knowledge of malware samples.

In response to the limitation of signature-based malware detection, various dynamic malware detection methods have been put forward [7]. Dynamic malware detection analyzes the sample behaviors during execution and generally called behavior-based malware detection. Behavior-based malware detection methods include virtual machine and function call monitoring, information flow tracking, and dynamic binary instrumentation. Windows Application Programming Interface (API) call graph-based method has been considered as a good prospect in behavior-based malware detection for a long time [8, 9].

Machine learning algorithms such as Decision Tree (DT), K-Nearest Neighbor (KNN), Naïve Bayes (NB), and Support Vector Machine (SVM) are commonly used in malware detection [10, 11]. The traditional machine learning algorithms can potentially learn the behavior features from the malware. Unfortunately, most machine learning algorithms' performance depends on the accuracy of the extracted features. In addition, it is often difficult to extract meaningful behavior features for improving malware detection performance. Moreover, feature processing requires expertise. Therefore, traditional machine learning algorithms are still somewhat unsatisfying for malware detection.

Deep learning is a branch of machine learning that attempts to learn high-level features directly from the original data. In short, deep learning advocates the end-to-end solution directly. It completely eliminates the whole process of large and challenging project phase. Deep learning is efficient to study high-level features of samples by means of multilayer deep architecture, and it has been widely used in image processing, visual recognition, object detection, etc. [12–17].

This paper introduces a method to protect IoT devices from being attacked by local computers. In this paper, we build a behavior-based deep learning framework (BDLF) which takes full advantage of Stacked AutoEncoders (SAEs) and traditional machine learning algorithms for malware detection. SAEs is one of the deep learning models that consists of multiple layers of sparse AutoEncoders [18, 19]. We use SAEs model extracts high-level features from behavior graphs and then do classification by the added classifiers (i.e., DT, KNN, NB, and SVM). DT, KNN, NB, and SVM combine with the SAEs model, called SAE-DT, SAE-KNN, SAE-NB, and SAE-SVM, respectively. The proposed BDLF is implemented in cloud platform.

In short, the main contributes are as follows:

(1) In this paper, we construct a novel behavior-based deep learning framework called BDLF by combing SAEs model with behavior graphs of API calls for malware detection. The proposed BDLF aims to obtain deeper semantics in behavior graphs rather than previous API call sequences (e.g., n-gram).

(2) In the proposed BDLF, we investigate a deep learning model of SAEs to automatically acquire high-level representations of malware behaviors. Our experiment results demonstrate that our method can extract more meaningful abstract features and help to improve the average precision in malware detection.

The remainder of this paper is organized as follows. Section 2 introduces related work. Section 3 describes the proposed behavior-based deep learning framework. The evaluation and experiment results are presented in Section 4, which is followed by the conclusion and future work in Section 5.

2. Related Work

With more and more malware attacks and smart devices' connection in IoT environment, security is not a separate event [20–22]. It is necessary to detect local computers' attacks for weakening the threats to other smart devices in IoT environment.

Malware detection proves an effective way for preventing IoT threats. Jiawei et al. present a method for detecting malware in IoT environment [23]. They first convert the extracted binaries into images and then use the convolutional Neural Network (CNN) to detect malware. The experiment demonstrated that their method obtains a good performance in malware detection. Pa et al. analyze the IoT devices and identify four malware families in IoT environment. They propose an IoT honeypot and sandbox for analyzing attacks.

Malware samples usually achieve their intentions by performing malicious actions on operating system resources. In [24], the proposed behavior model captures the interactions between malware and operating system resources which consist of file, registry, process, and network. Sanjeev et al. [25] observe the actions that are correlated with file system, process, network, and memory.

Behavior-based malware detection has witnessed a shift towards API calls [26]. The pattern of API calls provides an excellent expression which helps to "understand malware samples better." API calls provide efficient information about the runtime activities of a malware sample. Wu et al. [27] transform API calls into regular expressions and then use these rules to detect malware when a similar regular expression appeared. Taejin et al. [28] convert API calls into the formatted codes and group the API data using an n-gram. Pratiksha et al. [29] recognize malware by using API calls and their frequencies. Sanjeev et al. [25] propose a frequency-centric model for feature construction by employing API calls and OS resources of malware and benign samples.

Remarkably, deep learning is being applied for malware feature extraction and detection in recent years. Wenyi et al. [30] propose a deep learning architecture with the input rests on a sequence of API call events and null-terminated objects. Bojan et al. [31] use the Convolutional and Recurrent Network to analyze API call sequences in malware classification. Razvan et al. [32] explore a few variants of Echo State Networks (ESNs) and Recurrent Neural Networks (RNNs) to predict next API call. Omid E. et al. [33] extract unigrams (1-gram) API call and create an invariant compact representation of the malware behavior by using a Deep Belief Network (DBN). Wookhyun et al. [34] present a deep Recurrent Neural Network (RNN) to deal with the sequence of API calls. William et al. [12] design a deep learning architecture using SAEs model. The proposed architecture is

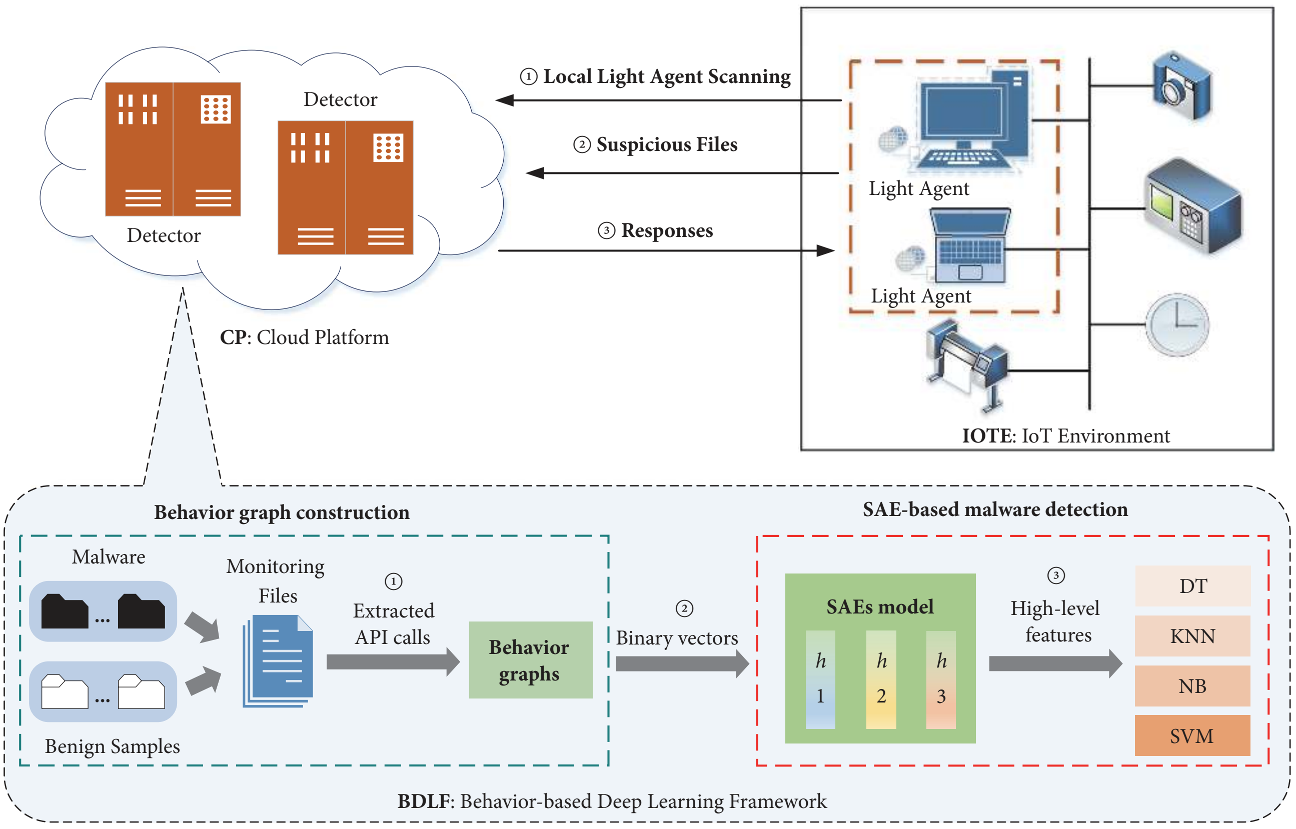

Figure 1: System overview.

based on the API calls extracted from the Portable Executable (PE) files.

Previous works have shown that different strategies can be used to build the patterns of API calls. However, the methods using API calls and their frequencies or API call fragments are limited. Ammar Ahmed E. et al. [35] demonstrate that combined API calls and their parameters raise the malware detection accuracy rather than considered API calls separately. In their study, each malware is represented as an API call graph by integrating API calls and operating system resources. They first extract API calls and their parameters through preprocessing and then use the proposed API call construction algorithm to build integrating API call graph. At last, they calculate the similarity between different graphs to identify the input sample.

Different from the previous works, the proposed BDLF is a combined approach using behavior graphs of API calls and SAEs model. Our approach aims to capture the high-level malicious behaviors for improving malware detection in IoT environment.

3. Behavior-Based Deep Learning Framework

We in this section elaborate the proposed BDLF. The proposed BDLF consists of two modules: behavior graph construction and SAE-based malware detection.

3.1. System Overview. The overview of our proposed system is displayed in Figure 1. The proposed system is composed of IoT environment (IoTE) and cloud platform (CP) module. The IoTE module consists of local computers and other smart devices. The proposed BDLF is implemented in CP's detectors, which is the main module for behavior construction and malware detection. In the proposed BDLF, each program is represented by a behavior graph which consists of many API call graphs. API call graph integrates API calls with operating system resources. After the behavior graphs are constructed, CP transforms the behavior features into binary vectors and then uses these vectors as input to the SAEs. There are 3 layers in the proposed SAEs model. The architecture of the SAEs is

$$h_1(6{,}000)\text{-}h_2(2{,}000)\text{-}h_3(500), \qquad (1)$$

and the last hidden layer's data are used as the input to the added classifiers (i.e., DT, KNN, NB, and SVM). The aim of the proposed BDLF is to learn the semantics of the high-level malicious behaviors and detect malware effectively. Specifically, the purpose and functionality of each component are described as follows.

(1) IoTE module refers to an IoT environment. The local computer contains an installed light agent which is responsible for collecting runtime activities. In this module, computers transfer the scanning information or suspicious files which are newly installed to CP and receive responses from CP.

Table 1: Operating system resource types and API calls.

Operating System Resource Types	Lists of API Calls
Service	OpenSCManager, OpenService, StartService
Process	NtOpenSection, ZwMapViewOfSection, NtFreeVirtualMemory, NtCreateSection, CreateProcessInternal, ExitProcess,
Filesystem	NtCreateFile, NtReadFile, NtSetInformationFile, NtOpenFile, NtWriteFile, DeviceIoControl, CreateDirectory, DeleteFile, FindFirstFile, NtDeviceIoControlFile, NtQueryInformationFile
Registry	RegOpenKey, RegSetValue, RegCloseKey, RegDeleteValue, RegQueryValue, RegCreateKey, NtOpenKey, NtQueryValueKey, RegEnumValue, RegEnumKey, NtQueryKey, RegQueryInfoKey
Synchronization	NtCreateMutant, NtOpenMutant
Network	WSAStartup, getaddrinfo
System	NtDelayExecution, FindWindow, SetWindowsHook, RemoveDirectory, GetSystemMetrics, LookupPrivilegeValue

```
(1) NtCreateFile,0x000000f8,. . . \nso1.tmp
(2) DeleteFile,. . . \nso1.tmp
(3) NtCreateFile,0x000000f8,. . . \Trojan-Downloader.Win32.Zlob.bcl
(4) NtQueryInformationFile,0x000000f8
(5) NtReadFile,0x000000f8
(6) NtReadFile,0x000000f8
(7) NtCreateFile,0x000000ec,. . . \nsi2.tmp
(8) NtSetInformationFile,0x000000f8
```

Box 1: Sample malware execution trace.

(2) CP provides an unlimited storage space. Detectors in CP are responsible for detecting scanning data or files received from IoTE. For scanning information, CP constructs behavior graphs and then transforms the API call graphs into binary vectors which are used as input to SAEs models for malware detection. For suspicious files, CP executes samples in Cuckoo Sandbox and then extracts API calls from sandbox's monitoring files. After that, CP manages the monitoring files the same way as the scanning information. After the detection, CP gives feedback to IoTE.

3.2. Behavior Graph Construction. The actions in behavior-based malware detection must only include security-critical operations and related independent operations [36]. We considered the actions performed on operating system resources which include seven types such as service, process, file system, registry, synchronization, network, and system. An action contains a set of operations which correspond to a set of related API calls [37–39]. We list some relationships between operating system resource types and some API calls in Table 1.

API calls listed in Table 1 easily happen in benign samples. However, the combination of these API calls may lead to malicious purpose with elaborate design. We propose the behavior graphs of API calls on malware. The proposed API call graphs are designed to learn malicious behaviors from the combination of API calls. Box 1 represents a code fragment of the malware:

(i) create nso1.tmp and then delete (in line (1), (2), respectively).

(ii) create Trojan-Downloader.Win32.Zlob.bcl and obtain its information; after that, read and set the file information of the Trojan-Downloader.Win32.Zlob.bcl (delete, rename, or change attributes, in line (3), (4), (5), (6), (8), respectively).

(iii) create nsi2.tmp (in line (7)).

Figure 2 represents three features (API call graphs) extracted from code fragment which is shown in Box 1. We construct extracted features by grouping related API calls which belong to the same operating system resource type. For example, the feature sets

{*NtCreateFile*, *DeleteFile*},

{*NtCreateFile*, *NtQueryInformationFile*, *NtReadFile*, *NtSetInformationFile*},

{*NtCreateFile*}

are performed on file resource nso1.tmp, Trojan-Downloader.Win32.Zlob.bcl, and nsi2.tmp, respectively. The first API call graph contains NtCreateFile and DeleteFile. NtCreateFile has two arguments: X_1 (X_1 = 0x000000f8) and X_2 (X_2 = . . . \nso1.tmp). DeleteFile has one argument X_3 (X_3 = . . . \nso1.tmp). The label $X_2 = X_3$ denotes that the second argument of NtCreateFile has the same value as the value of DeleteFile. The second API call graph contains NtCreateFile, NtQueryInformationFile, NtReadFile, and NtSetInformationFile. NtCreateFile has two

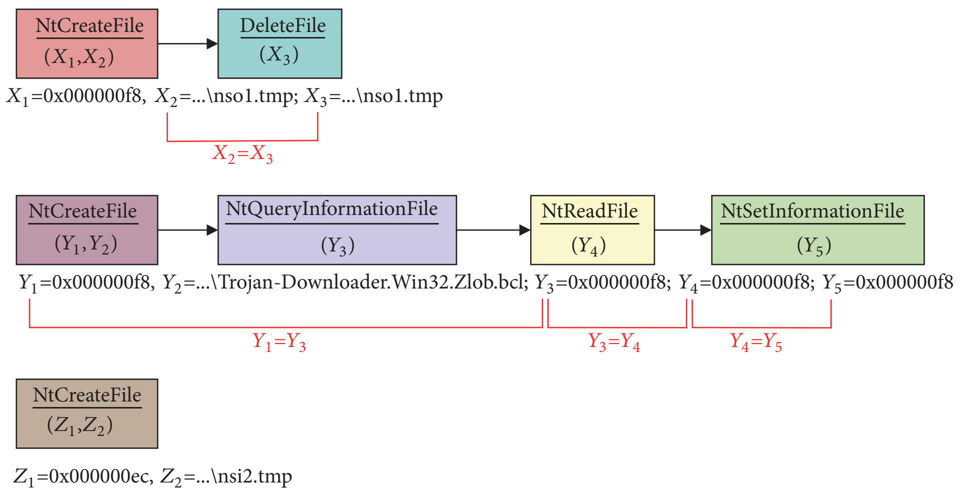
NtCreateFile
(X_1,X_2)
DeleteFile
(X_3)
X_1=0x000000f8, X_2=...\nso1.tmp; X_3=...\nso1.tmp
$X_2=X_3$
NtCreateFile
(Y_1,Y_2)
NtQueryInformationFile
(Y_3)
NtReadFile
(Y_4)
NtSetInformationFile
(Y_5)
Y_1=0x000000f8, Y_2=...\Trojan-Downloader.Win32.Zlob.bcl; Y_3=0x000000f8; Y_4=0x000000f8; Y_5=0x000000f8
$Y_1=Y_3$
$Y_3=Y_4$
$Y_4=Y_5$
NtCreateFile
(Z_1,Z_2)
Z_1=0x000000ec, Z_2=...\nsi2.tmp

Figure 2: Extracted malware features.

arguments: Y_1 (Y_1 = 0x000000f8) and Y_2 (Y_2 = ...\Trojan-Downloader.Win32.Zlob.bcl). NtQueryInformationFile has one argument of Y_3 (Y_3 = 0x000000f8). NtReadFile has one argument of Y_4 (Y_4 = 0x000000f8). NtSetInformationFile has one argument of Y_5 (Y_5 = 0x000000f8). The same as the labels $Y_1 = Y_3$, $Y_3 = Y_4$, and $Y_4 = Y_5$, $Y_1 = Y_3$ indicates that the first value of NtCreateFile has the same value as the value of NtQueryInformationFile. The third API call graph has only one node of NtCreateFile which has two arguments of Z_1 (Z_1 = 0x000000ec) and Z_2 (Z_2 = ...\nsi2.tmp).

Our proposed API call graph is a directed acyclic graph where nodes stand for either an API call or an operating system resource and edges represent some types of dependence. We define the proposed API call graph as

$$G = \left(V, E, f, A_V, A_E, \sum\right). \tag{2}$$

In the API call graph G: V stands for a set of nodes, $E \subseteq V \times V$ represents a set of edges, and f is a function that maps nodes V to API calls or operating system resources in the alphabet set $\sum$. Furthermore, each node in V and edge in E has its attribute which can be represented as A_V and A_E, respectively.

The proposed system monitors the API calls and their parameters to recognize malicious behaviors and has the following rules to identify hostile attacks.

API calls and their extension functions perform the same operation, and this kind of "sibling manipulation" leads to identical features. For example, whenever there is a need to open the registry, the open operations can be expressed as RegOpenKeyExA, RegOpenKeyExW, or other forms of expressions; we identify different forms of expressions as the same operation in our proposed system. In addition, the same API call which results in identical features is performed continuously more than two times, which we regard as one operation. In Box 1, the API call NtReadFile is performed twice on Trojan-Downloader.Win32.Zlob.bcl, and the API call graph is built as a standalone implementation of NtReadFile:

{*NtCreateFile*, *NtQueryInformationFile*, *NtReadFile*, *NtSetInformationFile*}.

The proposed API call graphs do not consider the order of the behaviors. Malware may perform malicious behaviors in totally different orders. The behaviors described in Figure 2:

{*NtCreateFile*, *DeleteFile*},

{*NtCreateFile*, *NtQueryInformationFile*, *NtReadFile*, *NtSetInformationFile*},

{*NtCreateFile*},

are considered identical to

{*NtCreateFile*, *DeleteFile*},

{*NtCreateFile*},

{*NtCreateFile*, *NtQueryInformationFile*, *NtReadFile*, *NtSetInformationFile*}.

Moreover, we use the API calls and operating system resource instances to identify API call graphs. From the example represented in Figure 2 we can see that the operating system resource is used to identify related operations rather than the feature vectors. This is because malware samples inclined to use random file names or other values every time when they are executed.

3.3. SAE-Based Malware Detection. Before using the behavior graphs of API calls as input to SAEs model, we transform these features into binary vectors. We employ one-hot encoding to identify unique behavior for every API call graph G. Let n be the number of the extracted API call graphs in the dataset Δ. API call graphs constructed in dataset Δ are denoted by binary feature vectors:

$$\Delta_{G_n} = (v_1, v_2, \dots, v_l, \dots, v_n) \tag{3}$$

where v_l represents the lth API call graph in dataset Δ. In our proposed system, the behavior graph of sample P_j can be represented as P_{jG_n}. Sample P_j is represented as (P_{jG_n}, C_i),

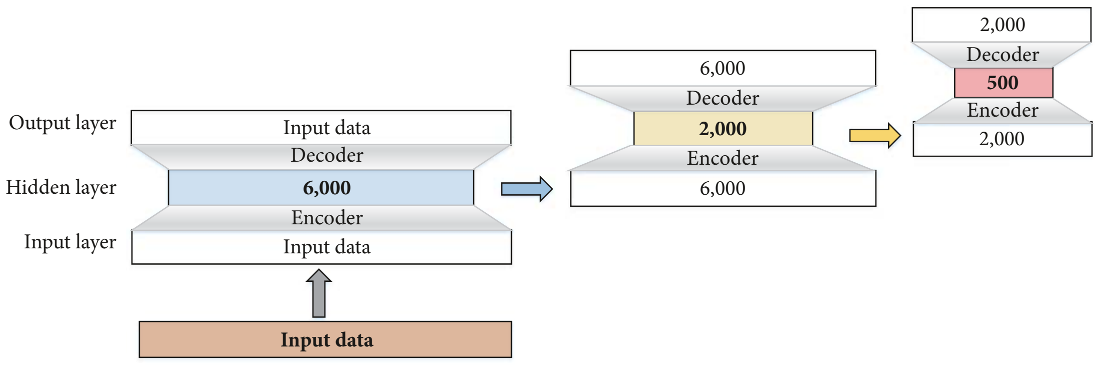

FIGURE 3: SAEs model in malware detection.

where C_i is the class label the sample P_j belongs to. There are two designated class labels associated with the proposed BDLF with C_1 representing the class of malware and C_2 representing the class of benign sample.

API call graphs in a sample P_j are then transformed into binary vectors and the behavior graph of sample P_j can be represented as

$$P_{jG_n} = \left(v_{j1}, v_{j2}, \ldots, v_{jl}, \ldots, v_{jn}\right). \quad (4)$$

Here $v_{jl} = \{0, 1\}$; if the sample P_j contains the API call graph v_l, $v_{jl} = 1$; otherwise $v_{jl} = 0$.

In order to build a deep neural network, we apply SAEs model which consists of multiple layers of sparse AutoEncoders to extract features [40, 41]. An AutoEncoder (AE) has three layers: input layer, hidden layer, and output layer. The hidden layer is located between the input layer and the output layer. An AE tries to use the encoder to map the input data into a hidden layer and use the decoder to map the hidden layer's data into an output layer, so as the output is similar to the input values. In short, an AE attempts to learn the sparse representation of the input and reconstruct the input data.

Figure 3 depicts the proposed SAEs model which contains 3 layers. In our approach, the proposed SAEs model consists of 3 hidden layers:

$$h_1(6{,}000)\text{-}h_2(2{,}000)\text{-}h_3(500). \quad (5)$$

Different hidden layers are trained one by one from bottom to top. In the proposed SAEs model, the first layer receives 11,164-sized original input data and trains simply as an AE. After training is completed in an AE, the hidden layer h_1 of 6,000-sized features generated in the first hidden layer is used as the input to a new AE which is added on top of the current AE. The new AE obtains the current AE's output as its input and trained similarity. Generally, the kth hidden layer's data are used as the input of the $(k + 1)th$ layer and trained simply as an AE. Finally, the last hidden layer's output is the entire SAEs model's output.

When all the training layers finished, the SAEs model converts the 11,164-sized original features into 500-sized high-level features. These 500-sized high-level features are regarded as the new presentations of an executable program file. The proposed SAEs model aims to reduce the number of the features and describe the features in a compact high-level expression.

Algorithm 1 describes a deep learning model which includes SAEs and some added classifiers for malware detection. Line (2) describes the input data activation[0] for each sample in Δ. Once the first layer in line (4) is pretrained, it can be used as an input to the next AE. We fine-tune the deep neural network after being pretrained in line (6) and put final layer h's activation to the added classifier to line (9). In line (9), $i = 1$ represents the classifier of DT, $i = 2$ represents the classifier of KNN, $i = 3$ represents the classifier of NB, and $i = 4$ represents the classifier of SVM. Line (10) and line (12) train the added classifiers and output the class label C_{out} (malware or benign sample). In our experiment, h equals 3, n equals 11,164, and m equals 4.

4. Evaluation and Experiment Results

In this section, we first explain the dataset we used for evaluation and evaluation method. To evaluate the effectiveness of our method, we then compare the proposed BDLF with some shallow models which consists of DT, KNN, NB, and SVM. Furthermore, we compare our method with other deep learning methods.

4.1. Dataset and Evaluation Method. We conduct the evaluation with a dataset containing 1760 samples, where 880 are malware samples, and the other 880 are benign samples. The malware samples are collected from VX Heaven. We analyze malware and benign samples in Cuckoo Sandbox. In our experiments, we use k-fold cross-validation method [42] in malware detection. In k-fold cross-validation, the original dataset is randomly divided into k equal-sized parts. We use 10-fold cross-validation in malware detection. For 10-fold cross-validation, we use 1584 samples for training and 176 samples for testing in each experiment.

We evaluate the proposed malware detection method by using the *F*1-*Score*. *F*1-*Score* is the weighting-harmonic-mean of the *Precision* and the *Recall*. Given the notions of true positive *TP* (the positive sample is correctly identified as the positive sample), true negative *TN* (the negative sample is correctly identified as the negative sample), false

```
Input: $\Delta$ including malware and benign samples $(P_1, P_2, \ldots, P_r)$
        sample $P_j$ under detection
Output: $C_{out}$  // the result of the detection
(1) Begin
        Construct binary feature vectors $P_{jG_n} = (v_{j1}, v_{j2}, \ldots, v_{jl}, \ldots, v_{jn})$
(2)     Activation[0]= $\{P_{1G_n}, P_{2G_n}, \ldots, P_{rG_n}\}$
(3)     For $k = 1$ to $h$ do
(4)       Train AE[$k$] use the activation AE[$k-1$] as the input and train $kth$ hidden
(5)       layer's parameters
(6)       Fine tune the neural network
(7)     End
(8)     For $i = 1$ to $m$ do  // $i$ represents different classifier
(9)       Add the classifier $i$ to the top layer of the SAEs model
(10)      Train the added classifier
(11)    End
(12)    Output the class label $C_{out}$
(13) End
```

Algorithm 1: Behavior-based deep learning model in malware detection.

Table 2: Precision, Recall, and F1-score.

Method	Precision	Recall	F1-Score
DT	0.968	0.967	0.968
SAE-DT	**0.986**	0.992	**0.989**
KNN	0.975	0.975	0.975
SAE-KNN	0.981	0.993	0.987
NB	0.911	0.906	0.905
SAE-NB	0.948	0.999	0.973
SVM	0.975	0.975	0.975
SAE-SVM	0.960	0.999	0.980

positive FP (the negative sample is incorrectly identified as the positive sample), and false negative FN (the positive sample is incorrectly identified as the negative sample), the *Precision* (P), *Recall* (R), and $F1$-*Score* are defined as follows.

$$P = \frac{TP}{TP + FP} \quad (6)$$

$$R = \frac{TP}{TP + FN} \quad (7)$$

$$F1\text{-}Score = \frac{2 * P * R}{P + R} \quad (8)$$

4.2. Experiment and Evaluation Results. In this section, based on the dataset introduced in Section 4.1, we evaluate the experiments in two aspects: shallow models and deep learning models. We conduct eight experiments. The experiments include some shallow models and SAE-based deep learning models.

The shallow models select behavior features by IG and then use these features to predict the labels of the samples. IG is used to denote information exchange and select certain properties [43]. The measurement criterion in IG is how much information the feature can bring to the system. The more information the feature brings, the more important it is. We describe IG in our previous work [11]. Shallow models include DT, KNN, NB, and SVM.

We train deep leaning models which include the proposed SAE-DT, SAE-KNN, SAE-NB, and SAE-SVM. 3 hidden layers' deep learning model

$$h_1(6{,}000)\text{-}h_2(2{,}000)\text{-}h_3(500) \quad (9)$$

is implemented on Keras. We feed our 11,164 features to SAEs model and convert them to 500-sized features. The batch size for the deep leaning models is 1,000. The SAE-based systems (SAE-DT, SAE-KNN, SAE-NB, and SAE-SVM) are trained with 100 epochs.

The average *Precision*, *Recall*, and $F1$-*Score* are shown in Table 2. It can be observed from Table 2 that the $F1$-*Score* of the SAE-based methods outperformed the other shallow methods. The best performance of the detection is that of the SAE-DT model. In the proposed SAE-DT model, the *Precision* is as high as 98.6%. The high performances were obtained from the SAE-DT, SAE-KNN, SAE-NB, and SAE-SVM model, which indicate that the features learned from the SAEs model help to improve the performance compared with traditional classification.

In addition, we compared our best $F1$-*Score* with previous works. William et al. [12] design a deep learning

Table 3: The comparison between our study and other deep learning methods.

Method	Features	Machine Learning Model	Precision	Recall	F1-Score
William et al. [12]	API calls	Stacked AutoEncoders	0.955	0.958	0.956
Zhenlong et al. [44]	Permissions, Sensitive APIs, App actions	Deep Belief Networks	0.968	0.968	0.968
Toshiki et al. [45]	Malware communication	Recursive Neural Network	0.976	0.962	0.969
Proposed SAE-DT	Behavior graphs	Stacked AutoEncoders	0.986	0.992	0.989

framework using the SAEs for intelligent malware detection based on API calls. The experiment results on their testing dataset demonstrate their proposed deep learning method achieves 95.5% detection precision. Zhenlong et al. [44] build an Android malware detection engine (DroidDetector) based on Deep Belief Networks (DBN). The proposed DroidDetector can achieve 96.8% detection precision by analyzing the features of required permissions, sensitive APIs, and dynamic behaviors (13 app actions). Toshiki et al. [45] focus on studying the similarity of data structure between malware communications and applying Recursive Neural Network (RNN) for malware analysis. Their proposed method achieves 97.6% detection precision. Our proposed BDLF based on SAEs and behavior graphs achieves 98.6% detection precision. It can be seen from Table 3 that our proposed SAE-DT improves the performance in malware detection. It is meaningful for mining the deep semantic relationships in behavior graphs.

5. Conclusion and Future Work

In this paper, we build a novel behavior-based deep learning malware detection framework in IoT environment for malware detection. By combining behaviors and Stack AutoEncoder, we obtain optimal detection performance. The experimental results in Section 4 demonstrate that SAE-based models can learn deeper abstract semantics features and help to improve the average precision of the detection by 1.5%. We are hopeful that additional works in SAEs model can be applied in malware detection and classification.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (Grant no. 61601041), the Fundamental Research Funds for the Central Universities (2018RC55), and the Beijing Talents Foundation (Grant no. 2017000020124G062).

References

[1] "Internet Security Threat Report," 2016, https://www.symantec.com/content/dam/symantec/docs/security-center/archives/istr-16-april-volume-21-en.pdf.

[2] H. Sun, X. Wang, R. Buyya, and J. Su, "CloudEyes: Cloud-based malware detection with reversible sketch for resource-constrained internet of things (IoT) devices," *Software: Practice and Experience*, vol. 47, no. 3, pp. 421–441, 2017.

[3] M. Alhanahnah, Q. Lin, Q. Yan, N. Zhang, and Z. Chen, "Efficient signature generation for classifying cross-architecture IoT malware," in *Proceedings of the 6th IEEE Conference on Communications and Network Security, CNS 2018*, Beijing, China, June 2018.

[4] S. Sharmeen, S. Huda, J. H. Abawajy, W. N. Ismail, and M. M. Hassan, "Malware threats and detection for industrial mobile-IoT networks," *IEEE Access*, vol. 6, pp. 15941–15957, 2018.

[5] S. Cesare, Y. Xiang, and W. Zhou, "Control flow-based malware variant detection," *IEEE Transactions on Dependable and Secure Computing*, vol. 11, no. 4, pp. 304–317, 2014.

[6] H. S. Galal, Y. B. Mahdy, and M. A. Atiea, "Behavior-based features model for malware detection," *Journal in Computer Virology and Hacking Techniques*, vol. 12, no. 2, pp. 59–67, 2016.

[7] A. Kharraz, A. Sajjad, C. Mulliner, W. Robertson, and K. Engin, "UNVEIL: a large-scale, automated approach to detecting ransomware," in *Proceedings of the USENIX Security Symposium*, pp. 757–772, 2016.

[8] A. A. E. Elhadi, M. A. Maarof, and B. I. A. Barry, "Improving the detection of malware behaviour using simplified data dependent API call graph," *International Journal of Security and Its Applications*, vol. 7, no. 5, pp. 29–42, 2013.

[9] B. S. Abhishek and B. A. Prakash, "Graphs for malware detection: the next frontier," in *Proceedings of the 13th International Workshop on Mining and Learning with Graphs (MLG)*, 2017.

[10] M. Fan, J. Liu, X. Luo et al., "Android malware familial classification and representative sample selection via frequent subgraph analysis," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 8, pp. 1890–1905, 2018.

[11] Z. Lin, X. Fei, S. Yi, Y. Ma, C.-C. Xing, and J. Huang, "A secure encryption-based malware detection system," *KSII Transactions on Internet and Information Systems*, vol. 12, no. 4, pp. 1799–1818, 2018.

[12] H. William, C. Lingwei, H. Shifu, Y. Yanfang, and L. Xin, "DL4MD: A deep learning framework for intelligent malware detection," in *Proceedings of the International Conference on Data Mining (DMIN)*, p. 61, The Steering Committee of the World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2016.

[13] O. Ronneberger, P. Fischer, and T. Brox, "U-net: convolutional networks for biomedical image segmentation," in *Proceedings of the International Conference on Medical Image Computing*

and Computer-Assisted Intervention (MICCAI '15), vol. 9351 of *Lecture Notes in Computer Science*, pp. 234–241, Springer, Cham, Switzerland, November 2015.

[14] W. Yang, Q. Liu, S. Wang et al., "Down image recognition based on deep convolutional neural network," *Information Processing in Agriculture*, vol. 5, no. 2, pp. 246–252, 2018.

[15] J. Donahue, L. A. Hendricks, S. Guadarrama et al., "Long-term recurrent convolutional networks for visual recognition and description," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015*, pp. 2625–2634, USA, June 2015.

[16] A. Voulodimos, N. Doulamis, A. Doulamis, and E. Protopapadakis, "Deep learning for computer vision: a brief review," *Computational Intelligence and Neuroscience*, vol. 2018, Article ID 7068349, 13 pages, 2018.

[17] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems*, pp. 91–99, 2015.

[18] E. Protopapadakis, A. Voulodimos, A. Doulamis, N. Doulamis, D. Dres, and M. Bimpas, "Stacked autoencoders for outlier detection in over-the-horizon radar signals," *Computational Intelligence and Neuroscience*, vol. 2017, Article ID 5891417, 11 pages, 2017.

[19] L. Vareka and P. Mautner, "Stacked autoencoders for the P300 component detection," *Frontiers in Neuroscience*, vol. 11, p. 302, 2017.

[20] S. Sicari, A. Rizzardi, L. A. Grieco, and A. Coen-Porisini, "Security, privacy and trust in Internet of Things: the road ahead," *Computer Networks*, vol. 76, pp. 146–164, 2015.

[21] S. Singh and N. Singh, "Internet of Things (IoT): Security challenges, business opportunities & reference architecture for E-commerce," in *Proceedings of the 1st International Conference on Green Computing and Internet of Things, ICGCIoT 2015*, pp. 1577–1581, IEEE, Noida, India, October 2015.

[22] L. Atzori, A. Iera, and G. Morabito, "The internet of things: a survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[23] J. Su, V. D. Vasconcellos, S. Prasad, S. Daniele, Y. Feng, and K. Sakurai, "Lightweight classification of IoT malware based on image recognition," in *Proceedings of the 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, pp. 664–669, Tokyo, Japan, July 2018.

[24] M. Chandramohan, H. B. K. Tan, and L. K. Shar, "Scalable malware clustering through coarse-grained behavior modeling," in *Proceedings of the 20th ACM SIGSOFT International Symposium on the Foundations of Software Engineering, FSE 2012*, p. 27, ACM, New York, NY, USA, November 2012.

[25] S. Das, Y. Liu, W. Zhang, and M. Chandramohan, "Semantics-based online malware detection: Towards efficient real-time protection against malware," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 2, pp. 289–302, 2016.

[26] Y. Ki, E. Kim, and H. K. Kim, "A novel approach to detect malware based on API call sequence analysis," *International Journal of Distributed Sensor Networks*, vol. 2015, no. 6, Article ID 659101, 9 pages, 2015.

[27] W. Liu, P. Ren, K. Liu, and H.-X. Duan, "Behavior-based malware analysis and detection," in *Proceedings of the 1st International Workshop on Complexity and Data Mining (IWCDM '11)*, pp. 39–42, IEEE, September 2011.

[28] T. Lee, B. Choi, Y. Shin, and J. Kwak, "Automatic malware mutant detection and group classification based on the n-gram and clustering coefficient," *The Journal of Supercomputing*, pp. 1–15, 2015.

[29] P. Natani and D. Vidyarthi, "Malware detection using API function frequency with ensemble based classifier," in *Proceedings of the International Symposium on Security in Computing and Communication*, vol. 377, pp. 378–388, Springer, Berlin, Germany, 2013.

[30] W. Huang and J. W. Stokes, "MtNet: a multi-task neural network for dynamic malware classification," in *Proceedings of the International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, vol. 9721 of *Lecture Notes in Computer Science*, pp. 399–418, Springer International Publishing, Cham, Switzerland, 2016.

[31] B. Kolosnjaji, A. Zarras, G. Webster, and C. Eckert, "Deep learning for classification of malware system call sequences," in *Proceedings of the Australasian Joint Conference on Artificial Intelligence*, Lecture Notes in Comput. Sci., pp. 137–149, Springer, Cham, Switzerland, 2016.

[32] R. Pascanu, J. W. Stokes, H. Sanossian, M. Marinescu, and A. Thomas, "Malware classification with recurrent networks," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1916–1920, IEEE, Australia, April 2014.

[33] O. E. David and N. S. Netanyahu, "DeepSign: deep learning for automatic malware signature generation and classification," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN '15)*, pp. 1–8, July 2015.

[34] J. Wookhyun, K. Sangwon, and C. Sangyong, "Poster: deep learning for zero-day flash malware detection," in *Proceedings of the 36th IEEE Symposium on Security and Privacy*, 2015.

[35] A. A. E. Elhadi, M. A. Maarof, B. I. A. Barry, and H. Hamza, "Enhancing the detection of metamorphic malware using call graphs," *Computers & Security*, vol. 46, pp. 62–78, 2014.

[36] P. M. Comparetti, G. Salvaneschi, E. Kirda, C. Kolbitsch, C. Kruegel, and S. Zanero, "Effective and efficient malware detection at the end host," in *Proceedings of the USENIX Security Symposium*, vol. 4, pp. 351–366, 2009.

[37] M. Chandramohan, H. B. K. Tan, L. C. Briand, L. K. Shar, and B. M. Padmanabhuni, "A scalable approach for malware detection through bounded feature space behavior modeling," in *Proceedings of the Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference*, pp. 312–322, November 2013.

[38] B. Ulrich, P. M. Comparetti, C. Hlauschek, K. Christopher, and E. Kirda, "Scalable, behavior-based malware clustering," in *NDSS*, vol. 9, pp. 8–11, 2009.

[39] D. Canali, A. Lanzi, D. Balzarotti, C. Kruegel, M. Christodorescu, and E. Kirda, "A quantitative study of accuracy in system call-based malware detection," in *Proceedings of the 21st International Symposium on Software Testing and Analysis, ISSTA 2012*, pp. 122–132, ACM, New York, NY, USA, July 2012.

[40] J. Yu, C. Hong, Y. Rui, and D. Tao, "Multitask autoencoder model for recovering human poses," *IEEE Transactions on Industrial Electronics*, vol. 65, no. 6, pp. 5060–5068, 2018.

[41] K. Zeng, J. Yu, R. Wang, C. Li, and D. Tao, "Coupled deep autoencoder for single image super-resolution," *IEEE Transactions on Cybernetics*, vol. 47, no. 1, pp. 27–37, 2017.

[42] Y. Zhang, S. Wang, P. Phillips, and G. Ji, "Binary PSO with mutation operator for feature selection using decision tree applied to spam detection," *Knowledge-Based Systems*, vol. 64, pp. 22–31, 2014.

[43] L. Wenke and D. Xiang, "Information-theoretic measures for anomaly detection," in *Proceedings of the Security and Privacy 2001 IEEE Symposium*, pp. 130–143, 2001.

[44] Z. Yuan, Y. Lu, and Y. Xue, "Droiddetector: android malware characterization and detection using deep learning," *Tsinghua Science and Technology*, vol. 21, no. 1, Article ID 7399288, pp. 114–123, 2016.

[45] T. Shibahara, T. Yagi, M. Akiyama, D. Chiba, and T. Yada, "Efficient dynamic malware analysis based on network behavior using deep learning," in *Proceedings of the GLOBECOM 2016 - 2016 IEEE Global Communications Conference*, pp. 1–7, Washington, DC, USA, December 2016.