API Call-Based Malware Classification Using Recurrent Neural Networks

Chen Li[1,*] and Junjun Zheng[2]

[1]*Department of Bioscience and Bioinformatics, Faculty of Computer Science and Systems Engineering, Kyushu Institute of Technology, Iizuka, Japan*
[2]*Department of Information Science and Engineering, Ritsumeikan University, Kusatsu, 525-8577, Japan*
E-mail: lichen7283@gmail.com; jzheng@asl.cs.ritsumei.ac.jp
[*]*Corresponding Author*

Received 09 January 2021; Accepted 25 February 2021;
Publication 25 May 2021

Abstract

Malicious software, called malware, can perform harmful actions on computer systems, which may cause economic damage and information leakage. Therefore, malware classification is meaningful and required to prevent malware attacks. Application programming interface (API) call sequences are easily observed and are good choices as features for malware classification. However, one of the main issues is how to generate a suitable feature for the algorithms of classification to achieve a high classification accuracy. Different malware sample brings API call sequence with different lengths, and these lengths may reach millions, which may cause computation cost and time complexities. Recurrent neural networks (RNNs) is one of the most versatile approaches to process time series data, which can be used to API call-based Malware calssification. In this paper, we propose a malware classification model with RNN, especially the long short-term memory (LSTM) and the gated recurrent unit (GRU), to classify variants of malware by using long-sequences of API calls. In numerical experiments, a benchmark dataset is used to illustrate the proposed approach and validate its accuracy. The

Journal of Cyber Security and Mobility, Vol. 10_3, 617–640.
doi: 10.13052/jcsm2245-1439.1036

numerical results show that the proposed RNN model works well on the malware classification.

Keywords: Malware classification, API call sequence, Recurrent neural network, Long short-term memory, Gated recurrent unit (GRU).

1 Introduction

With the rapid development of the Internet, the daily life of hundreds of millions of users is closely related to the Internet, such as online shopping, ordering meals, banking, etc. For example, Pandalabs has reported that there will be more than 50 billion devices connected to the Internet by 2020 [1]. Despite making our lives more convenient, the Internet makes us face the risk of being attacked, such as viruses, worms, and trojan horses.

Malicious software, or malware, can perform harmful actions on servers, computer systems, and mobile devices, to gain unauthorized access to destroy the system and steal data. Malware can not only maliciously expose users' private information, but also may pose security threats to the economy, resulting in undesired losses such as information leakage and economic damage. Therefore, great efforts are required to prevent malware attacks.

In general, there are two main approaches for malware classification, static-based approach [2, 3] and dynamic-based approach [4, 5]. For the static-based malware classification approach, the executable file is examined on the structure without viewing the actual instructions. For example, the opcode sequence is one of the malware features, which is usually used for static analysis of malware [6]. However, the information contained in opcode sequences is quite limited, and the cost of obtaining and executing the opcode is quite high. Moreover, the static-based approach is difficult to distinguish the type of malware by only observing the fixed features, since there are a large number of variants made every moment. For the dynamic-based malware classification approach, malware is executed and identified without the need for reverse engineering of malware.

A well-known approach for dynamic-based malware classification is extracting features from binary files, which can be transformed as the input of the Convolutional Neural Network (CNN). However, the information contained in the binary data is mainly spatial, which may lack temporal information of malware. Application Programming Interfaces (APIs) contain temporal features of malware, which can be also used for dynamic malware classification [7]. Since the malware calls the APIs provided by the operating

system (OS) to execute their malicious tasks, and API call sequences can be observed according to the dynamic analysis using some automated malware analysis systems such as the Cuckoo Sandbox and Alkanet tracer system. The observed API call sequences seem to be of a good choice as features for malware classification. However, one of the main issues is how to generate a suitable feature for the algorithms of classification to achieve a high classification accuracy, since different malware sample brings API call sequence with different lengths, and these lengths may reach millions resulting in difficult computation and increased time complexity.

A neural network is a popular mathematical approach, which can automatically find an approximate function with the given input. Recently, a number of tasks have been done using RNN, such as image caption generation [8], speech recognition [9] and language modeling [10]. Recurrent Neural Network (RNN) is one of the most versatile approaches to sequence-based classification [11]. Since there exists a hidden state in RNN, the prior information in the sequence can be memorized and updated [12]. Unfortunately, conventional RNN may cause the gradient vanishing problem when processing long sequences. To resolve the gradient vanishing problem, two popular RNN variations are proposed, which are called Long Short-term Memory (LSTM) [13, 14] and Gated Recurrent Unit (GRU) [15]. For the LSTM and GRU models, the long-term dependencies in sequences can be effectively learned with a memory cell and gated units. The GRU controls the flow of information like an LSTM unit, but without having to use the memory cell. In other words, the GRU exposes the full hidden content without any control. Besides, the performance of GRU is comparable to LSTM, but the computational efficiency is higher since the gated units are simpler. In this paper, we propose a novel malware classification model with RNN, especially LSTM and GRU, to clarify which one is better for malware classification with long-sequences of API calls.

The main contributions of this paper are as follows:

1. We use an RNN model, especially LSTM and GRU models, to classify malware families considering API call information.
2. A preprocessing *algorithm* is proposed to reduce the noise of the long sequences of API calls and to improve the classification accuracy.
3. Numerical experiments demonstrate several evaluation indicators to compare the classification accuracy.

The remainder of the paper is organized as follows. Section 2 gives an overview of related works. Section 3 briefly introduces the conventional RNN

model, and its two variants, named LSTM and GRU, respectively. Section 4 demonstrates the proposed model. In Section 5, the experimental results are exhibited. Finally, the paper is concluded in Section 6.

2 Related Works

In this section, the past literature on malware classification is overviewed. Specifically, the static-based and dynamic-based malware classification approaches are surveyed in Sections 2.1 and 2.2 respectively.

2.1 Static-based Malware Classification

For the static analysis techniques for malware classification, there is no need to execute the malware in a controlled environment. In general, the static-based approach usually integrates machine learning algorithms in the classifier. For example, the executable file is first executed by the reverse engineer. Then, the signature, which is extracted from the malware source code, is compared with a regularly updated database.

Schultz et al. [16] designed a framework and used naive Bayes to classify malicious code statically. In the work, they extracted the features of malware by using binary profiling, string sequences, and hex dumps, and they analyzed the entire set of malicious executables instead of only boot-sector viruses, or only Win32 binaries. Kolter et al. [17] used overlapping four-byte sequences as features to improve the accuracy of classification. They presented empirical results from an extensive study of inductive methods for detecting malicious executables in the wild. Shankarapani et al. [18] proposed two general malware classification methods; static analyzer for vicious executables and malware examiner using disassembled code, which used static API calls and assembly calls for analysis, respectively. The experiment showed that the assembly call-based method was superior to the API call-based method. Raff et al. [19] proposed a feedforward neural network to classify malicious executables from raw byte sequences. The solution avoided a number of the issues with the more common byte n-gram approach, such as brittle features and over-focusing on the PE-Header as important information.

However, almost all the above approaches are difficult to distinguish the malware families by only observing the fixed features, since there are a large number of variants made every moment. Moreover, the computation cost of the static-based analysis techniques is very high, such as the byte n-gram method, which is difficult to be extended in a large domain.

2.2 Dynamic-based Malware Classification

The dynamic-based malware classification approach executes the executable file and identifies features without the need for reverse engineering of malware, such as memory writes, system calls, and API calls.

For example, Gregio et al. [20] proposed a dynamic model capture malware behavior with memory writes, which is a certain subset of instructions writes to memory during program execution. In 2015, Canzanese et al. [21] analyzed system calls to monitor executing processes to classify malware that evade traditional defense. The system monitored executing processes to identify compromised hosts in production environments. Also, Canzanese et al. [22] presented a malware analysis system, which is used to classify malicious processes at run-time on production hosts. Different to the other works, the analysis system does not require the use of specialized analysis environments. Eskandari et al. [23] proposed a binary classification approach by applying n-gram to distinguish malware binaries from API calls. In general, API call sequences contian more dynamical information of malware than the static information, such as signatures. Ahmed et al. [24] extracted spatial and temporal information from API calls and used the extracted features to classify malware. They applied statistical analysis to extract the arguments and return values from API calls as the spatial features. Moreover, they applied a Markov chain to extract the temporal information from the transition matrix by using API calls. Hansen et al. [25] and Qiao et al. [26] also proposed a malware classification system with API call-related features. Hansen et al. used API call sequences as the features for malware family classification, which are transformed from the dynamic analysis results of the Cuckoo sandbox analysis. Different from Hansen's work, Qiao et al. conducted the malware classification by using the frequency of the API call sequences. The API call sequences were obtained from CWSandbox and Cuckoo Sandbox.

In general, the API calls often appear in the form of sequences, and the lengths of the sequences can vary from one to millions, which made the feature extraction difficult. To improve the accuracy of the malware classification from API call sequences, many researchers considered using the models of deep learning, such as the RNN model. RNN has demonstrated a powerful ability in processing time series sequences, especially in natural language processing. For example, Yazı et al. [27] presented a binary classification for malware families with a single-layer LSTM model. The LSTM model can distinguish whether an API call sequence is a malware or not, and the API

call dataset was described in [28]. Also, Catak et al. [29] detected API calls from the Windows Operating system, which is a new dataset and does not exist in this domain before. In the work, they proposed a single layer LSTM and a two-layer LSTM to classify malware families, respectively. Tobiyama et al. [30] proposed a malware process detection method based on process behavior in possible infected terminals. Specifically, they first proposed an LSTM to extract the sequence features from API calls, and then the extracted features are feed into a convolutional neural network (CNN). The CNN continuously extracted and classified the features for a binary classifier.

Most of the above LSTM models considered the malware classification problem as a binary classification, which can only classify the executable files or the identified features as malware. In this paper, we propose an RNN model, especially LSTM and GRU, to classify malware into different families, such as viruses, worms, and trojans. Also, we present to use API calls as the raw data for the RNN model. To improve the malware classification ability and reduce the noise of the RNN model, we propose a simple but effective algorithm to extract the effective information from the API call sequences.

3 Preliminaries

In this section, we briefly introduce the conventional RNN model and two well-known variants, called LSTM cell and GRU cell, respectively.

3.1 Conventional RNN Model

RNN is a kind of artificial neural network to model sequential data. In general, RNN maintains a state, which can be updated at each time step, so that the previous information can be memorized and utilized for current prediction [12]. Figure 1 shows an example of the basic RNN model. Blank circle represents a state and gray circle represents a input. In this work, the input is the API call sequences, and the output is the malware family labels, such as viruses and worms. If we unfolded the RNN according to time steps, we can represent RNN as the network like the one shown in right part of Figure 1. $\boldsymbol{x}_t$, $\boldsymbol{y}_t$ and $\boldsymbol{h}_t$ be the input, output and hidden state of the RNN model at t-th time step, and $\boldsymbol{U}$, $\boldsymbol{W}$ and $\boldsymbol{V}$ represent the weight matrices. Then, the hidden state $\boldsymbol{h}_t$ is updated based on the previous $\boldsymbol{h}_{t-1}$ and the current input $\boldsymbol{x}_t$:

$$\boldsymbol{h}_t = f(\boldsymbol{U}\boldsymbol{x}_t + \boldsymbol{W}\boldsymbol{h}_{t-1}) \tag{1}$$

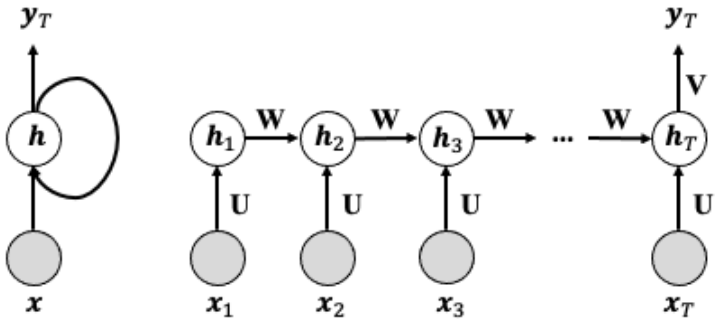

Figure 1 An example of the conventional RNN structure.

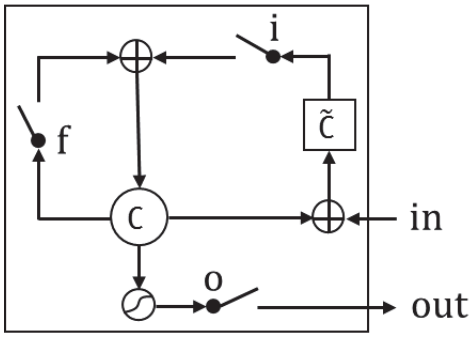

Figure 2 An example of the LSTM cell.

where f is the activation function, such as sigmoid, tanh, and ReLU function. The output $\boldsymbol{y}_T$ is computed by:

$$\boldsymbol{y}_T = \text{softmax}(\boldsymbol{V}\boldsymbol{h}_T) \tag{2}$$

3.2 Long Short-term Memory

LSTM [13] and GRU are the two variants of the basic RNN model, which are better at extracting long term dependencies of time series data. Figure 2 demonstrates the LSTM cell. In general, there exist three main gates in a general LSTM cell, which are called the input gate, forget gate, and output gate, respectively.

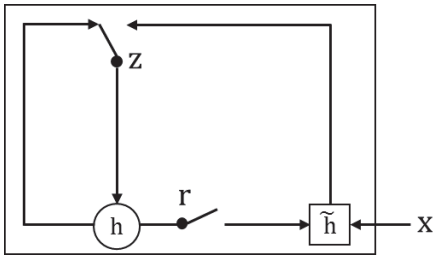

Figure 3 An example of the GRU cell.

The LSTM function is implemented by the following composite functions:

$$\boldsymbol{i}_t = \sigma\left(\boldsymbol{W}_{xi}\boldsymbol{x}_t + \boldsymbol{W}_{hi}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{ci}\boldsymbol{c}_{t-1} + \boldsymbol{b}_i\right) \tag{3}$$

$$\boldsymbol{f}_t = \sigma(\boldsymbol{W}_{xf}\boldsymbol{x}_t + \boldsymbol{W}_{hf}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{cf}\boldsymbol{c}_{t-1} + \boldsymbol{b}_f) \tag{4}$$

$$\boldsymbol{c}_t = \boldsymbol{f}_t\boldsymbol{c}_{t-1} + \boldsymbol{i}_t\tanh\left(\boldsymbol{W}_{xc}\boldsymbol{x}_t + \boldsymbol{W}_{hc}\boldsymbol{h}_{t-1} + \boldsymbol{b}_c\right) \tag{5}$$

$$\boldsymbol{o}_t = \sigma\left(\boldsymbol{W}_{xo}\boldsymbol{x}_t + \boldsymbol{W}_{ho}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{co}\boldsymbol{c}_{t-1} + \boldsymbol{b}_o\right) \tag{6}$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t\tanh(\boldsymbol{c}_t) \tag{7}$$

where $\boldsymbol{i}_t$, $\boldsymbol{f}_t$, $\boldsymbol{o}_t$, and $\boldsymbol{c}_t$ are the input gate, forget gate, output gate and cell content, respectively. Also, σ and tanh are the sigmoid and tanh function. $\boldsymbol{W}_{hi}$, $\boldsymbol{W}_{xo}$, $\boldsymbol{W}_{xi}$, $\boldsymbol{W}_{ci}$, $\boldsymbol{W}_{xf}$, $\boldsymbol{W}_{hf}$, and $\boldsymbol{W}_{xc}$ are the hidden-input gate matrix, input-output gate matrix, input gate matrix, cell-input matrix, input-forget gate matrix, hidden-forget gate matrix and input-cell gate matrix, respectively. $\boldsymbol{b_i}$, $\boldsymbol{b}_f$, $\boldsymbol{b}_c$ and $\boldsymbol{b}_o$ are the basis of input gate, forget gate, cell and output gate, respectively.

3.3 Gated Recurrent Unit

Different from the LSTM cell, the structure of GRU is relatively simple. Figure 3 shows the basic structure of a GRU model. In general, there are two main gates in a GRU [15] model, which are called the reset gate and update gate, respectively.

Define $\boldsymbol{r}_t$ and $\boldsymbol{z}_t$ represent the reset gate and update gate at t-th time step, respectively, which are computed as follows:

$$\boldsymbol{r}_t = \sigma(\boldsymbol{W}_r\boldsymbol{x}_t + \boldsymbol{U_r}\boldsymbol{h}_{t-1}) \tag{8}$$

$$\boldsymbol{z}_t = \sigma(\boldsymbol{W}_z\boldsymbol{x}_t + \boldsymbol{U_z}\boldsymbol{h}_{t-1}) \tag{9}$$

where $\boldsymbol{W}_r$, $\boldsymbol{U}_r$, $\boldsymbol{W}_z$ and $\boldsymbol{U}_z$ are weight matrices of the reset gate and update gate, respectively. Then, the hidden state $\boldsymbol{h}_t$ can be calculated as follows:

$$\boldsymbol{h}_t = \boldsymbol{z}_t \boldsymbol{h}_{t-1} + (1 - \boldsymbol{z}_t)\widetilde{\boldsymbol{h}_t} \tag{10}$$

$$\widetilde{\boldsymbol{h}_t} = \tanh(\boldsymbol{W}_r \boldsymbol{x}_t + \boldsymbol{U}_r(\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1})) \tag{11}$$

where $\odot$ represents the Hadamard product.

4 API Call Based Recurrent Neural Networks

The RNN-based malware classification model is introduced in this section. In general, we first describe the creation of the malware dataset, and then, we propose a novel preprocessing algorithm for the dataset to reduce the noises of the API call sequences. Finally, we generate the RNN model to classify the malware families.

4.1 Extraction of the API Call Sequences

The malware family samples are used to generate the malware executable repository (MER). Then, the generated API call sequences are sent to Cuckoo Sandbox for the dynamic analysis. Cuckoo Sandbox is a well-known framework for malware dynamic analysis, which can generate analysis reports for each malware sample. The generated analysis reports contain static and dynamic analysis results, such as hash values. In this paper, we only use the API call sequences as the input of the RNN model. On the other hand, the VirusTotal service API is used to generate malware family labels. The malware family labels and API call sequences together form the malware dataset.

For a better understanding, Figure 4 demonstrates the process of dataset creation. The created dataset contains a series of $(\mathrm{label},\ \mathrm{seqence})$ pairs, where the label represents the type of malware, while the $\mathrm{sequence}$ contains a series of names of the executed API calls.

Table 1 shows some examples of the $(\mathrm{label},\ \mathrm{seqence})$ pairs. L_i represents malware family label. For example, L_i is a kind of worms, viruses, trojan, downloader, backdoor, dropper, spyware, and adware. S_i reprensents the name of API calls, such as "_exception_" and "regclosekey". In the example, it is observable that the sequence of L_1 type calls the API file S_1 repeatedly. Also, the L_2 and L_3 repeatedly call the sub-sequence $\{S_1, S_2\}$ and $\{S_1, S_2, S_3\}$, respectively. However, the repeatedly called API sub-sequences

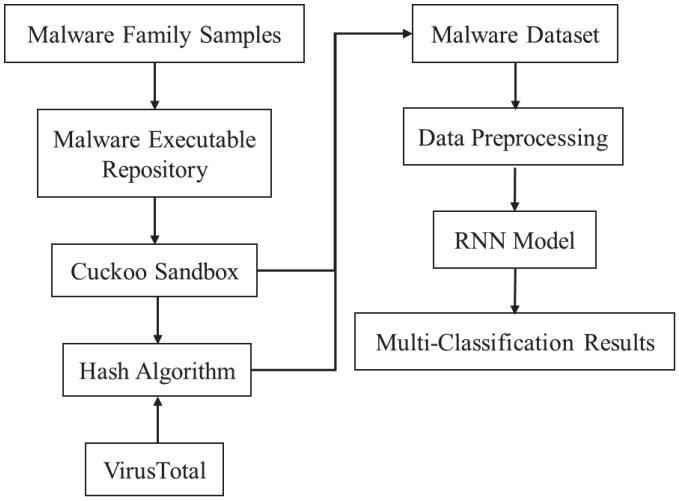

Figure 4 The process of the API call generation and classification.

Table 1 An example of the (label, sequence) pairs in the dataset

label	sequence
L_1	$S_1,\ S_1,\ S_1,\ S_1, \ldots$
L_2	$S_1,\ S_2,\ S_1,\ S_2, \ldots$
L_3	$S_1,\ S_2,\ S_3,\ S_1, S_2,\ S_3, \ldots$
...	...

have little effect on the classification results. On the contrary, the repeated API calls or the sub-sequences may increase the noise of the RNN model.

4.2 Data Preprocessing

To reduce the noise of the RNN model, we propose a simple but effective algorithm to preprocess the original API call sequences. For example, since there exists the repeated API S_1 in the API sequence of $\{S_1,\ S_1,\ S_1,\ S_1, \ldots\}$, we remove the continuously same API S_1. Besides, for the API sequences $\{S_1,\ S_2,\ S_1,\ S_2, \ldots\}$ and $\{S_1,\ S_2,\ S_3,\ S_1, S_2,\ S_3, \ldots\}$, there also exist the same API sub-sequences $\{S_1,\ S_2\}$ and $\{S_1,\ S_2,\ S_3\}$, we delete these continuously same sub-sequences, respectively. The algorithm of sequence preprocessing is illustrated in Algorithm 1. After data preprocessing, data noise and the lengths of

API call sequences are reduced. The algorithm result str as the input is feed into the RNN model.

Algorithm 1 Removal of continuously same API call.

Input: The original API call sequence seq.

Output: Sequence str after removing the duplicate pattern.

$s_1 = [\mathrm{i}[0]$ **for** $\{\mathrm{i}$ in $\mathrm{groupby}(seq)\}]$
$str =$ ''.join(s_1)
index = []
for i in range $(\mathrm{len}(str))$:
  if $str[\mathrm{i}] == str[\mathrm{i}+2]$ **and** $str[\mathrm{i}+1] == str[\mathrm{i}+3]$:
    index.extend($[\mathrm{i}+2, \mathrm{i}+3]$)
    i += 2
    s_2 = np.delete(str, index)
    $str =$ ''.join(s_2)
index = []
for i in range $(\mathrm{len}(str))$:
  if $str[\mathrm{i}] == str[\mathrm{i}+3]$ **and** $str[\mathrm{i}+1] == str[\mathrm{i}+4]$ **and** $str[\mathrm{i}+2] == str[\mathrm{i}+5]$:
    index.extend($[\mathrm{i}+3, \mathrm{i}+4, \mathrm{i}+5]$)
    i += 3
    s_3 = np.delete(str, index)
    $str =$ ''.join(s_3)
return str

4.3 RNN Model Description

Formally, given an API call sequence $\boldsymbol{X} = \{x_1, x_2, \ldots, x_m\}$, where x_i $(1 \leq i \leq m)$ is the index of an API call from the API dictionary with size v. Notice that m is very different since the length of the API call sequence can reach one to millions as we mentioned above. However, there exists a large number of redundant subsequences in the API call sequence, which have little effect on malware classification. To reduce the length of the sequence to fit the proposed LSTM model, we remove the continuously same API from the sequence. Although the removal operation is a simple method, it can effectively reduce the length of sequences and improve the accuracy of classification. Let T be the length of the LSTM model, then the sequence

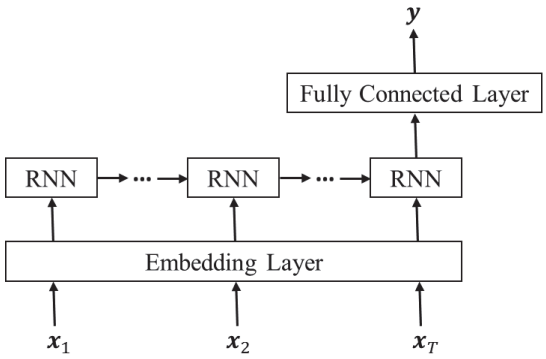

Figure 5 The architecture of the proposed RNN model.

is padded or truncated to $\boldsymbol{X} = \{x_1, x_2, \ldots, x_T\}$. In our model, we use v-dimensional one-hot vector to represent x_i, which means $x_i \in \mathbb{R}^v$.

Figure 5 demonstrates the architecture of our proposed RNN model. The one-hot vector is high-dimensional and sparse, which means only one element is 1, and the rest are 0 in the one-hot vector. Then, the one-hot vector x_i as input to be feed into the embedding layer. The embedding layer uses a dense vector to represent the one-hot vector, which can reduce the dimension of the one-hot vector.

The output is defined by $\mathbf{y}$, where $\mathbf{y} = [y_1, y_2, \ldots, y_K]$ and $\boldsymbol{y} \in \mathbb{R}^v$. $\boldsymbol{y}$ is generated by the softmax function, which is a probability distribution over all *K* malware families. In other words, y_i represents the probability of malware family *i*, where $i = (1, 2, \ldots, K)$, and $\sum_{i=1}^{K} y_i = 1$.

In the training phase, backpropagation-through-time (BPTT) is often used for propagating gradients of errors. BPTT is a variant of the backpropagation approach for training feedforward neural networks. Error is computed using cross-entropy loss function:

$$\text{Loss}(\boldsymbol{y}', \mathbf{y}) = \sum_{i=0}^{K} y_i' \log(y_i) \tag{12}$$

where $\boldsymbol{y}'$ is the true probability distribution, and $\mathbf{y}$ is the predicted probability distribution.

Besides, we apply Adagrad as the optimization method, which is a modified stochastic gradient descent method with a per-parameter learning rate. Also, the dropout technique is used as a regularization to alleviate overfitting.

5 Numerical Experiments

In this section, several extensive experiments are conducted to evaluate the effectiveness of our proposed model We evaluate some indications by using a benchmark dataset. Finally, we evaluate the performance of the proposed LSTM model and GRU model. All experiments are run in Keras 2.4.3 with Python 3.5.6.

5.1 Dataset

A benchmark dataset for malware classification is used in the experiment. The dataset contains 7,107 malware records for 8 different families of malware, which are represented by 0 to 7. The 8 families are worms, viruses, trojan, downloader, backdoor, dropper, spyware, and adware. Figure 6 shows the number of malware sequences for the 8 families, where the x-axis represents the eight malware families, and the y-axis is the number of corresponding sequences of each family.

Figure 7 shows the original lengths of API call sequences, where the x-axis and y-axis represent the lengths of API call sequences and the number of corresponding sequences, respectively.

Figure 8 demonstrates the lengths of API call sequences after the preprocessing operation.

It is observable that the lengths of most of the API call sequences are less than 400 after removing redundant API calls from the original sequences.

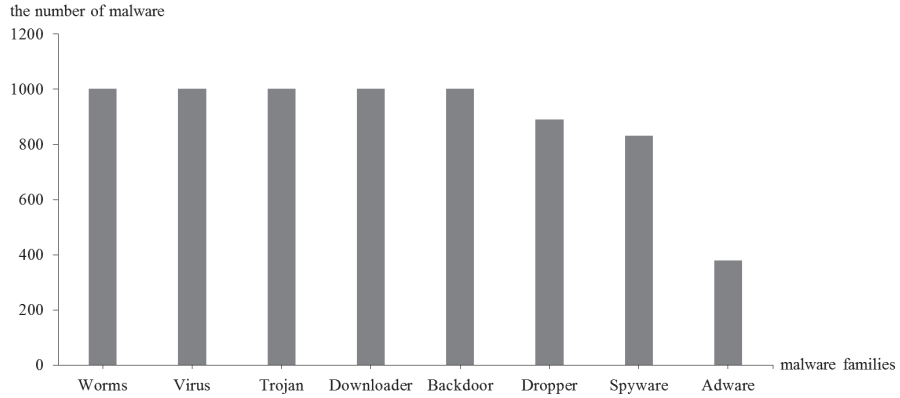

Figure 6 The number of malware sequences for the 8 families.

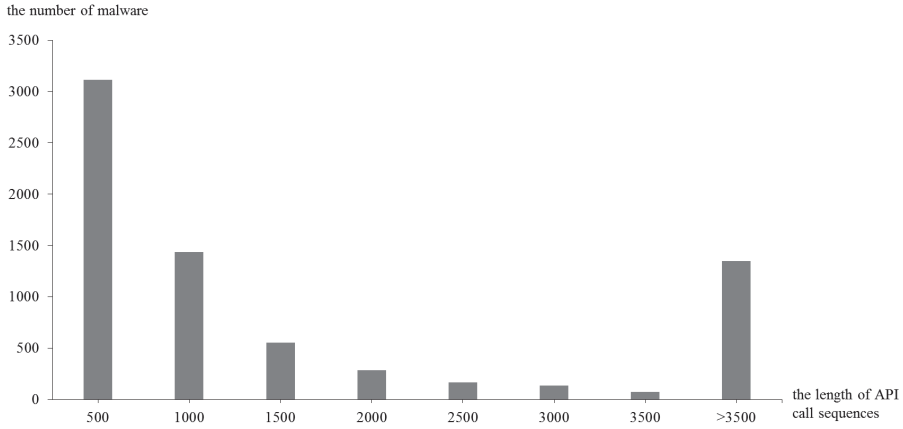

Figure 7 Length distribution of API call sequences before preprocessing.

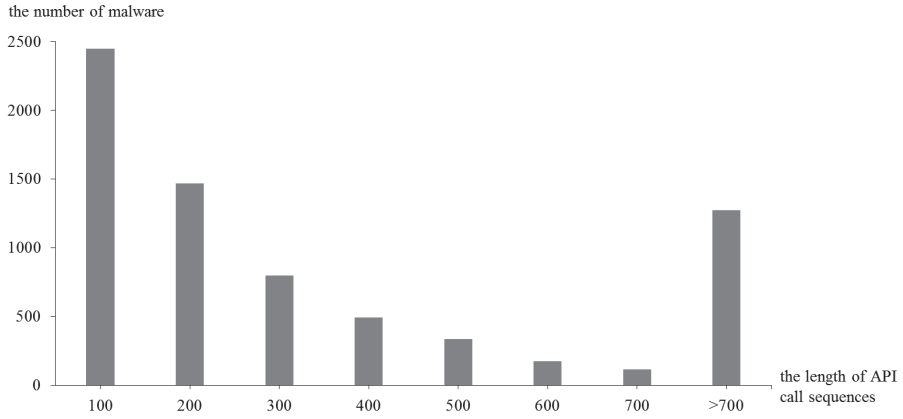

Figure 8 Length distribution of API call sequences after preprocessing.

Predicted

Actual		Positive	Negative
	Positive	TP	FN
	Negative	FP	TN

Figure 9 Confusion matrix for each of the eight malware families.

5.2 Performance Metrics

We calculate the accuracy, precision, recall, F1-score, Macro-average and weighted-average. Formally, define TP, TN, FP and FN as the true positives,

true negatives, false positives and false negatives, respectively. Figure 9 demonstrates the confusion matrix for each of the eight malware families.

Then, the accuracy, precision, recall and F1-score can be defined as

$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{13}$$

$$\text{precision} = \frac{TP}{TP + FP} \tag{14}$$

$$\text{recall} = \frac{TP + TN}{TP + FN} \tag{15}$$

$$\text{F1} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{16}$$

Besides, the Macro-average can be computed by averaging the precision, recall, and F1-score, respectively. Also, similar to the Macro-average, the weighted-average metric assigns weights to every malware family according to the number of samples of each family, hence the name.

5.3 Parameter Setting

We select a single hidden layer of the RNN model to classify the eight different types of malware. More specifically, we use the LSTM model and GRU model for the classification, respectively. Tables 2, 3 and Figures 10, 11 show the single hidden layer RNN model and the optimal hyperparameters of the RNN model, respectively.

Table 2 Optimal hyperparameters for the proposed LSTM model

Hyperparameters	Value
LSTM length	200
Embedding units	32
Length of hidden layer	32
Activation function	Sigmoid function
Kernel initializer	Random uniform
Dropout	0.5
Optimizer	Adam

Table 3 Optimal hyperparameters for the proposed GRU model

Hyperparameters	Value
GRU length	200
Embedding units	64
Length of hidden layer	64
Activation function	Sigmoid function
Kernel initializer	Random uniform
Dropout	0.5
Optimizer	Adam

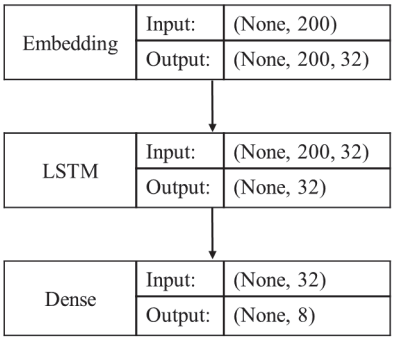

Figure 10 Single hidden layer LSTM classification model structure.

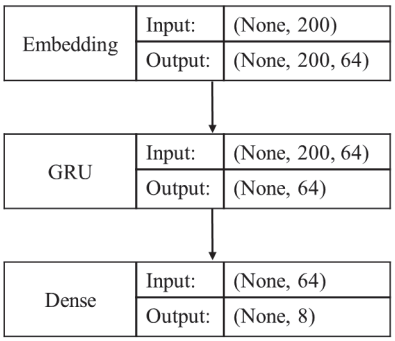

Figure 11 Single hidden layer GRU classification model structure.

5.4 Experimental Results

Figures 12–15 show the training loss, accuracy of the LSTM model and GRU model, respectively. Notice that the EarlyStopping approach is used to solve the overfitting issue. The analysis results of the testing phase are shown in

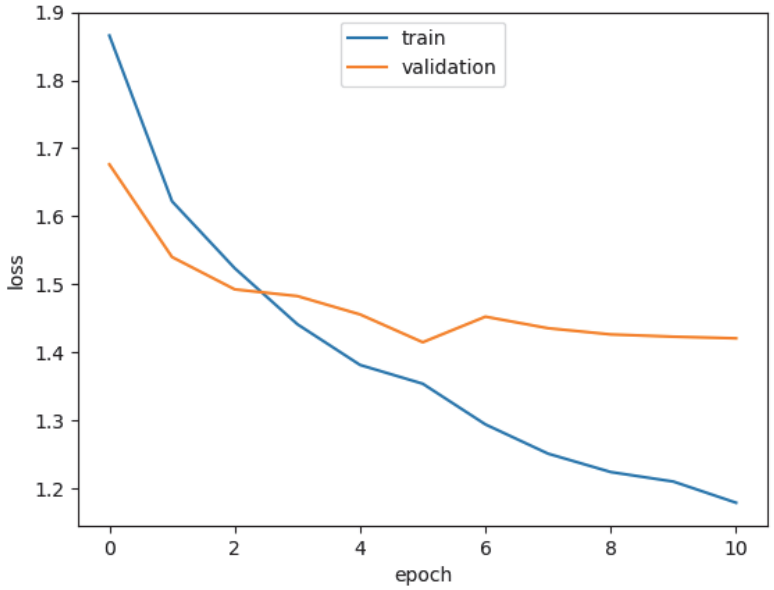

Figure 12 Training loss of the proposed LSTM model.

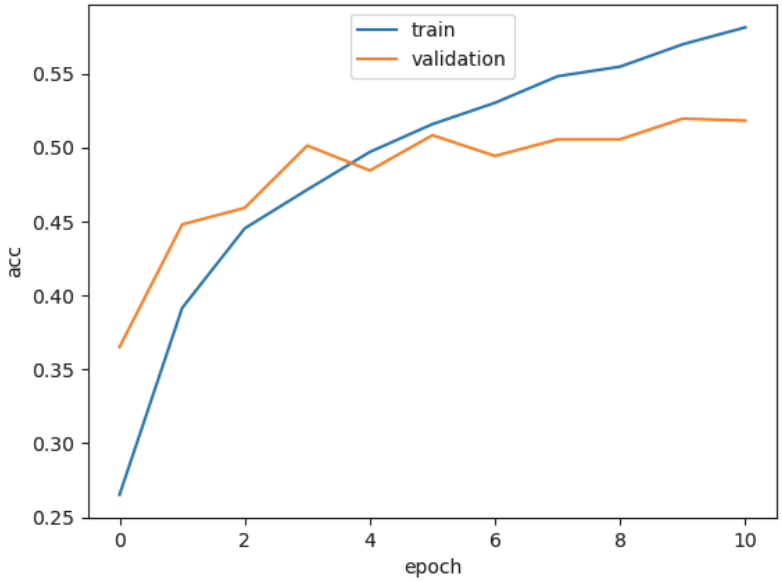

Figure 13 Training accuracy of the proposed LSTM model.

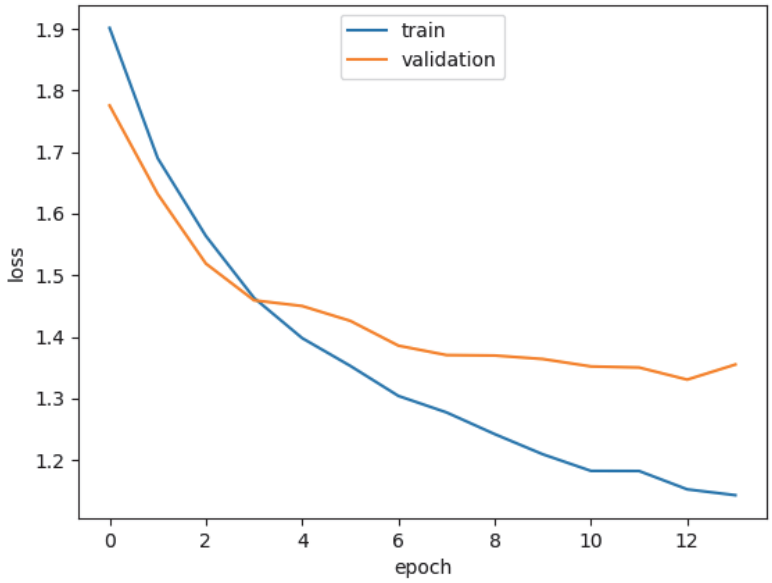

Figure 14 Training loss of the proposed GRU model.

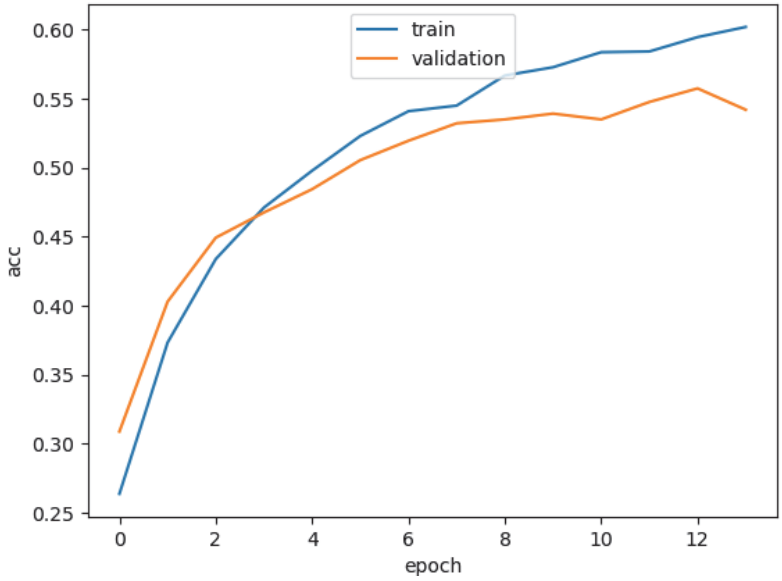

Figure 15 Training accuracy of the proposed GRU model.

Table 4 Classification results of the proposed LSTM model

	Precision		Recall		F1-score	
Malware Families	Case 1	Case 2	Case 1	Case 2	Case 1	Case 2
Adware	0.58	**0.77**	**0.95**	0.91	0.72	**0.83**
Backdoor	**0.60**	0.55	0.28	**0.51**	0.38	**0.53**
Downloader	0.65	**0.71**	0.52	**0.72**	0.58	**0.71**
Dropper	0.26	**0.62**	0.37	**0.51**	0.30	**0.56**
Spyware	0.14	**0.38**	0.24	**0.32**	0.17	**0.35**
Trojan	0.05	**0.30**	0.23	**0.38**	0.09	**0.33**
Virus	0.53	**0.72**	0.76	**0.76**	0.63	**0.74**
Worms	**0.47**	0.45	0.35	**0.52**	0.40	**0.48**
Accuracy	–	–	–	–	0.41	**0.55**
Macro-average	0.41	**0.56**	0.46	**0.58**	0.41	**0.57**
Weighted-average	0.50	**0.55**	0.41	**0.55**	0.43	**0.55**

Table 5 Classification results of the proposed GRU model

	Precision		Recall		F1-score	
Malware Families	Case 1	Case 2	Case 1	Case 2	Case 1	Case 2
Adware	0.64	**0.74**	0.88	**0.90**	0.74	**0.81**
Backdoor	**0.68**	0.40	0.39	**0.65**	0.49	**0.50**
Downloader	0.59	**0.64**	0.71	**0.74**	0.64	**0.69**
Dropper	0.40	**0.61**	**0.47**	0.46	0.44	**0.53**
Spyware	0.08	**0.36**	0.24	**0.38**	0.12	**0.37**
Trojan	0.36	**0.40**	0.25	**0.32**	0.29	**0.36**
Virus	0.63	**0.68**	**0.79**	0.76	0.70	**0.72**
Worms	0.43	**0.60**	0.45	**0.52**	0.44	**0.56**
Accuracy	–	–	–	–	0.48	**0.55**
Macro-average	0.48	**0.56**	0.52	**0.59**	0.48	**0.57**
Weighted-average	0.52	**0.55**	0.48	**0.55**	0.48	**0.54**

Tables 4 and 5. Notice that case 1 and case 2 represent the evaluation values of the original sequences and after preprocessing, respectively.

It is observable that the precision, recall and F1-score of the Spyware and Trojan families are quite lower than other malware families. One reason may be that the noise influence of these two families may cause low performance.

Also, the performance of the proposed LSTM and GRU model in case 2 are almost the same, and they are better than the performance of the model in case 1. Moreover, the performance of the proposed GRU model in case 1 is better than the proposed LSTM model in case 1. In other words, the API call-based LSTM model and GRU model work well on the malware classification.

6 Conclusions

Malware classification with time-series data, such as the API call sequences, is important to finance and the economy. RNN has achieved great success in prediction and classification with time-series data. In this paper, we proposed an RNN-based architecture to classify 8 malware families with API call sequences. More specifically, the LSTM cell and GRU cells were considered as the model for malware classification. In the experiment, a benchmark dataset was used to evaluate the classification accuracy, precision, recall, F1-score, Macro-average, and weighted-average. The analysis results illustrated that the proposed LSTM model and GRU model are effective in the API call-based malware classification.

In the future, we would like to compare the considered approaches in this paper with other classification methods. Another derection is to consider a hybrid neural network, such as the LSTM-CNN model, for the malware classification with API call sequences. Moreover, the attention mechanism is also needed to be considered to improve the accuracy of the classification.

References

[1] "Pandalabs quarterly report: Q2 2017," 2017. [Online]. Available: https://www.pandasecurity.com/mediacenter/src/uploads/2017/08/Pandalabs-2017-Q2-EN.pdf, accessed: 2020-07-27.

[2] A. Damodaran, F. D. Troia, C. A. Visaggio, T. H. Austin, and M. Stamp, "A comparison of static, dynamic, and hybrid analysis for malware detection," *J. Comput. Virol. Hacking Techn.,* vol. 13(1), pp. 1–12, Dec. 29, 2015.

[3] M. Ijaz, M. H. Durad, and M. Ismail, "Static and dynamic malware analysis using machine learning," *Proceedings of the 16th Int. Bhurban Conf. Appl. Sci.Technol. (IBCAST)*, pp. 793–806, Jan. 8–12, 2019.

[4] L. Nataraj, V. Yegneswaran, P. Porras, and J. Zhang, "A comparative assessment of malware classification using binary texture analysis and

dynamic analysis," *Proceedings of the 4th ACM Workshop Secur. Artif. Intell. AISec*, pp. 21–30, Oct. 2011.

[5] T. Shibahara, T. Yagi, M. Akiyama, D. Chiba, and T. Yada, "Efficient dynamic malware analysis based on network behavior using deep learning," *Proceedings of the 2016 IEEE Global Commun. Conf. (GLOBECOM)*, pp. 1–7, Dec. 4–8, 2016.

[6] W. Wong and M. Stamp, "Hunting for metamorphic engines," *J. Comput. Virol.*, vol. 2(3), pp. 211–229, 2006.

[7] M. K. Shankarapani, S. Ramamoorthy, R. S. Movva, S. Mukkamala, "Malware detection using assembly and API call sequences," *J. Comput. Virol.*, vol. 2(7), pp. 107–119, 2011.

[8] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," *Proceedings of the International Conference on Machine Learning*, pp. 2048–2057, 2015.

[9] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, and T. N Sainath, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29(6), pp. 82–97, 2012.

[10] T. Mikolov, M. Karafiat, L. Burget, J. Cernocky, and S. Khudanpur, "Recurrent neural network based language model," *In Interspeech*, vol. 2(3), 2010.

[11] I. Kwon, and E. G. Im, "Extracting the representative API call patterns of malware families using recurrent neural network," *Proceedings of the International Conference on Research in Adaptive and Convergent Systems*, pp. 202–207, 2017.

[12] C. Li, X. Zhang, M. Qaosar, S. Ahmed, K. Md. R. Alam, and Y. Morimoto, "Multi-factor based stock price prediction using hybrid neural networks with attention mechanism," *Proceedings of the 2019 IEEE International Conference on Cloud and Big Data Computing (CBDCom)*, pp. 961–966, Fukuoka, Japan, 2019.

[13] S. Hochreiter, and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9 (8), pp. 1735–1780, 1997.

[14] C. Li, Minjia He, Mahboob Qaosar, Saleh Ahmed, and Yasuhiko Morimoto, "Capturing temporal dynamics of users' preferences from purchase history big data for recommendation system," *Proceedings of the IEEE International Conference on Big Data 2018 (BigData2018)*, Seattle, WA, USA, USA, Dec. 2018.

[15] K. Cho, B. V. Merrienboer, D. Bahdanau, and B. Yoshua, "On the properties of neural machine translation: encoder–decoder approaches," *arXiv:1409.1259*, 2014.

[16] M. G. Schultz, E. Eskin, F. Zadok, and S. J. Stolfo, "Data mining methods for detection of new malicious executables," *Proceedings of the IEEE Symp. Secur. Privacy. S&P*, pp. 38–49, May, 2001.

[17] J. Z. Kolter, and M. A. Maloof, "Learning to detect malicious executables in the wild," *Proceedings of the ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining KDD*, pp. 470–478, 2004.

[18] M. K. Shankarapani, S. Ramamoorthy, R. S. Movva, and S. Mukkamala, "Malware detection using assembly and API call sequences," *J. Comput. Virology*, vol. 7(2), pp. 107–119, 2011.

[19] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. K. Nicholas, "Malware detection by eating a whole exe," *Proceedings of the Workshops 32nd AAAI Conf. Artif. Intell.*, pp. 268–276, 2018.

[20] A. R. A. Gregio, P. L. de Geus, C. Kruegel, and G. Vigna, "Tracking memory writes for malware classification and code reuse identification," *Proceeding of the Detection of Intrusions and Malware, and Vulnerability Assessment*, pp. 134–143, Berlin, Heidelberg, 2013.

[21] R. Canzanese, S. Mancoridis, and M. Kam, "System call-based detection of malicious processes," *Proceedings of the International conference on quality, reliability, and security (QRS)*, 2015.

[22] R. Canzanese, S. Mancoridis, and M. Kam, "Run-time classification of malicious processes using system call analysis," *Proceedings of the 10th International Conference on Malicious and Unwanted Software (MALWARE)*, pp. 21–28, 2015.

[23] M. Eskandari, Z. Khorshidpur, and S. Hashemi, "To incorporate sequential dynamic features in malware detection engines," *Proceedings of the Eur. Intell. Secur. Informat. Conf.*, pp. 46–52, Aug. 2012.

[24] F. Ahmed, H. Hameed, M. Z. Shafiq, and M. Farooq, "Using spatiotemporal information in api calls with machine learning algorithms for malware detection," *Proceedings of the 2nd ACM workshop on Security and artificial intell.*, pp. 55–62, 2009.

[25] S. S. Hansen, T. M. T. Larsen, M. Stevanovic, and J. M. Pedersen, "An approach for detection and family classification of malware based on behavioral analysis," *Proceedings of the International Conference on Computing, Networking and Communications (ICNC)*, pp. 1–5, 2016.

[26] Y. Qiao, J. He, Y. Yang, and L. Ji, "Analyzing malware by abstracting the frequent itemsets in API call sequences," *Proceedings of the 12th IEEE*

International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), pp. 265–270, 2013.

[27] A. F. Yazi, F. O. Catak, and E. Gul, "Classification of Metamorphic Malware with Deep Learning (LSTM)," *IEEE Signal Processing and Applications Conference*, 2019.

[28] F. O. Catak, and A. F. Yazi, "A benchmark API call dataset for windows PE malware classification," *CoRR*, 2019.

[29] F. O. Catak, A. F. Yazi, and O. Elezaj, "Deep learning based Sequential model for malware analysis using Windows exe API Calls," *PeerJ Computer Science*, vol. 6(81), July 27, 2020.

[30] S. Tobiyama, Y. Yamaguchi, H. Shimada, T. Ikuse, and T. Yagi, "Malware detection with deep neural network using process behavior," *Proceedings of the IEEE 40th Annu. Comput. Softw. Appl. Conf. (COMPSAC)*, pp. 577–582, Jun. 10–14, Atlanta, GA, USA, 2016.

Biographies

Chen Li received the B.S. degree in computer engineering from Qingdao College, Ocean University of China, in 2012, and the M.S. and D.Eng. degrees in engineering from Hiroshima University, Higashihiroshima, Japan, in 2016 and 2019, respectively. In 2019 and 2020, he was a Visiting Researcher with the Graduate School of Advanced Science and Engineering, Hiroshima University, Japan. Since 2021, he has been a post-doctoral researcher with the Department of Bioscience and Bioinformatics, Faculty of Computer Science and Systems Engineering, Kyushu Institute of Technology, Japan. His research interests include performance evaluation, data mining and deep learning.

Junjun Zheng received the B.S.E. degree in engineering from Fujian Normal University, Fuzhou, China, in 2010, and the M.S. and D.Eng. degrees in engineering from Hiroshima University, Higashihiroshima, Japan, in 2013 and 2016, respectively. In 2016 and 2017, he was a Visiting Researcher with the Department of Information Engineering, Graduate School of Engineering, Hiroshima University. Since 2018, he has been an Assistant Professor with the Department of Information Science and Engineering, Ritsumeikan University, Japan. His research interests include performance evaluation and dependable computing.

Zheng is a member of the Operations Research Society of Japan, the Reliability Engineering Association of Japan, the Institute of Electrical, Information and Communication Engineers, and the Institute of Electrical and Electronics Engineers.