國立政治大學資訊管理研究所

碩士學位論文

歸納惡意軟體特徵

Malware Family Characterization

指導教授：郁方 博士

研究生：劉其峰 撰

中 華 民 國 107 年 7 月

DOI:10.6814/THE.NCCU.MIS.025.2018.A05

Malware Family Characterization

Chi-Feng Liu

August 27, 2018

DOI:10.6814/THE.NCCU.MIS.025.2018.A05

Abstract

Nowadays, a massive amount of sensitive data which are accessible and connected through personal computers and cloud services attracts hackers to develop malicious software (malware) to steal them. Owing to the success of deep learning on image and language recognition, researchers direct security systems to analyze and identify malware with deep learning approaches. This paper addresses the problem of analyzing and identifying complex and unstructured malware behaviors by proposing a framework of combining unsupervised and supervised learning algorithms with a novel sequence-aware encoding method. Particularly, we adopt a hybrid GHSOM (the Growing Hierarchical Self-Organizing Map) algorithm to cluster and encode similar malware behavior sequences from system call sequences to clustering feature vectors. Then, a Recurrent Neural Network (RNN) is trained to detect malware and predict their corresponding malware families based on the sequence of the behavior vectors. Our experiments show that the accuracy rate can be up to 0.98 in malware detection and 0.719 in malware classification of an 18-category malware dataset.

DOI:10.6814/THE.NCCU.MIS.025.2018.A05

Contents

DOI:10.6814/THE.NCCU.MIS.025.2018.A05

List of Figures

DOI:10.6814/THE.NCCU.MIS.025.2018.A05

List of Tables

DOI:10.6814/THE.NCCU.MIS.025.2018.A05

1 Introduction

Malicious software, called malware, is the software with malicious functions. Malware is also known as computer virus after personal computers invaded people's homes. The massive sensitive personal data stored in personal computer attracts the hackers to steal them by using the sophisticated malicious software which is a significant threat. Malware may propagate to vulnerable computers, force them to execute malicious code, cause damages to the systems, or steal sensitive data. Malware is usually characterized by its functionalities, such as backdoor, bot, ransomware, phishing, worm, Trojan horse, and virus [3] and they could cause huge amounts of harm[4]. Therefore, detecting malware is an important and urgent problem in cybersecurity. Many researchers have already delved into this field and attempt to devise effective methods for detection[5]. The goal of this paper is to provide a novel framework for malware detection and classification.

In general, there are two mainstreams for malware detection: static analysis and dynamic analysis. In static analysis, malware features are extracted from malware executables or codes in a non-runtime environment. Static analysis tools, such as Manalyze[6], provide the extraction of the executable's headers and sections. However, static analysis is unable to extract the features that are only revealed in the runtime environment and are potentially obfuscated by the misleading malware code. In the dynamic studies, malware is run and monitored in a virtualized environment, and the features are collected from the obvious observations. Dynamic analysis performs in a different way, including tracking information flow, collecting hardware performance, and tracing system calls[3]. The functionality of a software program can be briefly determined by observing its system call trace which using only a subset of the system call trace, such as ignoring input arguments, ignoring returned data, and sampling the data[7]. Dynamic analysis tools can collect the invoked system calls in the virtual machine (VM) or the files modified by malware. Dynamic analysis is a time-consuming job, and sometimes malicious activities may not be observable. However, some research works[8] suggest that the data collected from dynamic analysis can lead to more accurate detection results than the data from

DOI:10.6814/THE.NCCU.MIS.025.2018.A05

static analysis.

This study establishes a malware detection and classification model based on the similar program behaviors. We observe that when malware does harm to the system, malware may need to change the system states (e.g., modify files, spawn processes, and perform interrupts) by invoking system calls, so we adopt the dynamic analysis approach to collect malware system call sequences. The recorded system calls are a huge number, so it is often split in n-gram as an interval behavior. Each program behaviors consists of a series of interval behaviors, one of which is considered as a basic program behavior. In order to analyze interval behaviors from system call sequences, we develop a two-stage encoding method. First, a system call frequency encoding method is developed to transform system call sequence to n-gram frequency encoding sequence to reduce data size. Then, the encoded interval behaviors are fed into the Growing Hierarchical Self-Organizing Map (GHSOM) algorithm for clustering analysis. Interval behaviors will be classified into different clusters with similar interval behavior. Second, we further develop a hierarchical SOM binary encoding method to transfer the multi-layered GHSOM clustering result to a one-dimension hierarchical SOM binary encoding sequences. In this way, the sequences are machine-operatable as the malware behavior sequences. We feed them into a Recurrent Neural Network (RNN) as the input for malware detection and malware family classification. Since each malware family has different purposes and functionalities, we anticipate their extracted execution sequences are distinguishable. By analyzing the execution sequence, we can distinguish malware and benign software and further classify malware into corresponding families.

In the literature, deep learning algorithms are widely adopted for malware detection and classification[9]. First, this study implements interval behavior learning using unsupervised clustering which use the GHSOM algorithm to classify the interval behaviors into different interval behavior clusters. Second, interval behavior sequence for malware detection and classification using Recurrent Neural Networks forward a series of malware behaviors to RNN model for predicting the malware family and detection malware. Unlike

DOI:10.6814/THE.NCCU.MIS.025.2018.A05

conventional researches, we not only propose a malware analysis framework by combining unsupervised learning and supervised learning but also design a sequence-aware encoding method. The contributions of this paper are as follow:

- We construct a hybrid behavior-based neural network framework for analyzing sequence data on the classification problem.
- We combine the unsupervised learning algorithm (GHSOM) and a supervised learning algorithm (LSTM) to increase the accuracy for malware detection and classification.
- We propose two-stage sequence encoding methods to compress the sequence size, preserve the sequential information and represent the hierarchical clustering results in machine-operatable style for further deep analysis.

The structure of the rest of this paper is prepared as follows. Section II is related work about the neural network, malware classification, and detection. In Section III, we introduce our methodology, the malicious behaviors sequence, the proposed behaviors clustering algorithm, and the RNN classifier. In Section IV, the experiment configuration and the results are illustrated. Section V concludes the paper, recaps the findings of malware classification, and discusses future work that can be surveyed further.

2 Related Work

2.1 Neural Network for Malware Classification and Detection

In 2016, Kolosnjaji et al.[10] construct a deep learning system to classify the system call sequences by combining a convolutional layer and recurrent layer in one neural network. Such a hybrid model performs better than the conventional Hidden Markov Model and Support Vector Machine. Toriyama et al.[11] conduct a malware infection detection system with two-stage deep NN, Convolutional Neural Network (CNN) and RNN. Features

DOI:10.6814/THE.NCCU.MIS.025.2018.A05

are extracted from the trained RNN, based on a language model with LSTM from a 5-minute API sequence. The feature images are then fed to the CNN. This method leads to the malware detection accuracy raising to 0.96. Rhode et al.[8] develop a malware prediction model based on RNN which reduces dynamic analysis time by only monitoring the activity data from ten machines within 20 seconds, and its detecting accuracy is around 0.96. Pascanu et al.[12] use a hybrid model which combine RNN with a standard classifier to improve the true positive rate. They build an API call language model by using RNN with API calls. The feature vectors generated by RNN are then classified by a multi-layer perception method. Chiu et al.[13] introduce a distributed GHSOM algorithm for malware clustering analysis which is able to identify malicious behaviors and generate the rule of each behaviors cluster.

2.2 Recurrent Neural Networks

Recurrent Neural Networks (RNN) [14][15] is a type of neural network which accepts time-sequential input data. Unlike feed-forward neural network, RNN uses their memory to "remember" the previous output state and resends it to the neuron. RNN is very powerful in analyzing the context and time-based data[16] but the exploding gradient problem and the vanishing gradient problem make it inadequate to train long time data.

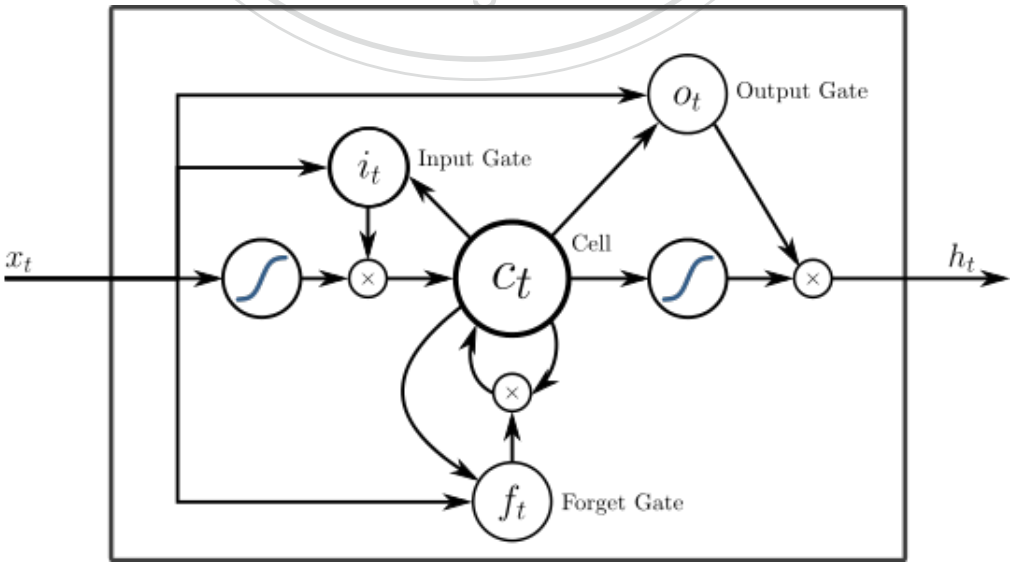

Figure 1: Long Short-Term Memory Model [1]

DOI:10.6814/THE.NCCU.MIS.025.2018.A05

To solve these problems, Long Short-Term Memory (LSTM)[14] is introduced, and it is capable of bridging time intervals in long steps. Each neuron in the LSTM has several gates that are used to remember or forget the previous neuron output. In LSTM (Fig. 1), the Forget Gate resets the cell states to zero or decreases the cell states slowly, which makes the LSTM cell easily carry an important message over long steps[17]. The Input Gate is to decide whether the current input and memory cell candidate are added to long-term memory by using a sigmoid function. The Output Gate acts the same as the Input Gate, but it works on the output value. Several models are developed on the top of RNN or LSTM, and Chung et al.[18] compares the difference between LSTM, gated recurrent unit (GRU), and traditional tanh unit in sequence modeling. RNN are now widely and effectively used in sequential data analysis such as language model[19], speech recognition[20], machine translation[21].

2.3 Unsupervised Learning Clustering

Many kinds of research analyze malware behaviors through the unsupervised clustering methods. However, the behavior data might be high dimensional or sequential, which is difficult to represent and learn. In 1990, the self-organizing feature maps (SOM) [22] was developed, and it is suitable to process high-dimension and multi-feature dataset with different data categories. Rauber et al. 2002 [23] propose the growing hierarchical self-organizing map (GHSOM) which automates the process of projecting the high-dimension data onto a hierarchical tree structure of SOM. It reveals the hierarchical relationship of the high-dimension data on SOM more easily. In 2014, Shi et al. [24] adopt such hierarchical SOM approach to distinguish the malware categories. They show that the structure of the hierarchical SOM is adequate to identify the fast-changing malware. Also, the proposed hierarchical SOM can grow a malware category tree by considering the similarity of malware. Shuwei et al.[25] use Shared Nearest Neighbor clustering algorithm and take the frequency of system calls as the input data. Then, it adopts the density-based spatial clustering of applications with noise (DBSCAN) to offer a simple and automated

DOI:10.6814/THE.NCCU.MIS.025.2018.A05

malware analysis.

In summary, it is a tendency that malware researches adopt NN or machine learning approach to analysis malware behavior because in general, the malware behavior is too complicated for human beings to analyze. Our proposed analysis framework leverages on both unsupervised and supervised learning algorithm to deal with malware analysis. Unlike the past researches (who use NN for encoding), we use an unsupervised method (GHSOM) and the proposed encoding method to automatically encode non-real-number sequence data (i.e., system call sequence) and preserve the sequence information. In such way, we can revisit the malicious cluster to understand the malicious behavior. Second, we adopt LSTM for the classification. The LSTM model can train the sequence information we left in the encoded sequence, which is more appropriate than the convolution model.

3 Methodology

The methodology to perform malware classification is illustrated in the section. It includes the system overview, the proposed two encoding methods for system call sequences, and the proposed combination of unsupervised learning clustering and recurrent neural network for malware detection and classification.

3.1 System Overview

Our process is displayed in figure 2. Software pool is a buffer zone where we store software and system calls recorded from executing software in the virtual machine environment. The recorded system calls are preprocessed to get interval behavior feature vectors which represent the behaviors of malware and benignware (benign software) in a fixed period. We use system call frequency encoding method to transfer system calls into interval behavior feature vectors. Each interval behavior feature vectors are then transferred to the unsupervised clustering algorithm model, GHSOM (The Growing Hierarchical Self-Organizing Map)[26], which is used to classify each interval behavior feature vectors into different clusters. Every behavior feature vector is corresponding to a cluster of behavior. In order

DOI:10.6814/THE.NCCU.MIS.025.2018.A05

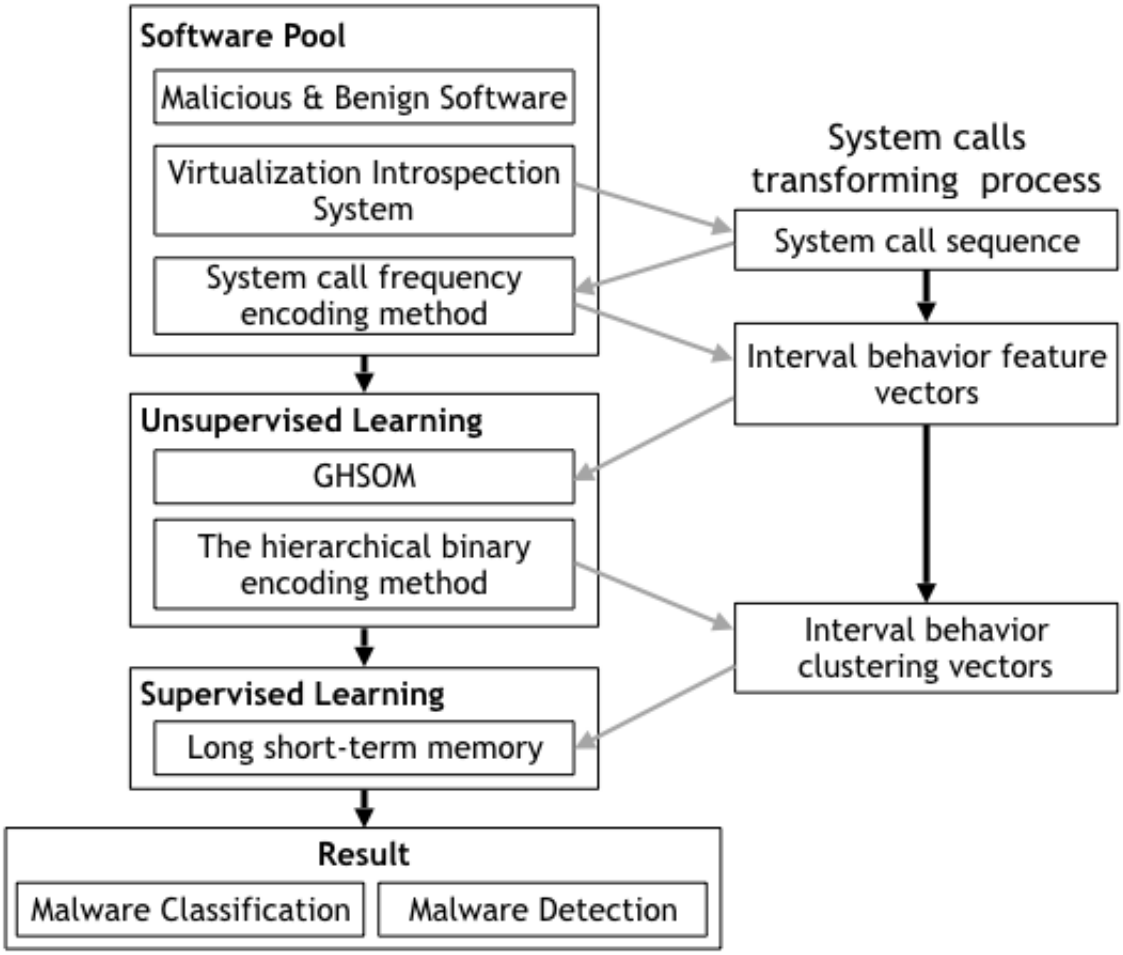

Figure 2: System overview

to describe the structure of GHSOM layers, each GHSOM clusters are transformed to behavior cluster vector by using the hierarchical SOM binary encoding method. Then, program behaviors sequence comprise behavior cluster vectors chronologically. The program behavior sequence is then fed to the recurrent neural network, which detects whether the software malicious or not and classifies malware into different families.

3.2 Software Pool

All programs and system call sequences are stored in the software pool. A system call is a computer program requests a service from the kernel of the operating system. We use Cuckoo Sandbox[27] , an open source automated malware analysis system and simulate the environment for executing software to retrieve system calls in runtime execution.

DOI:10.6814/THE.NCCU.MIS.025.2018.A05

VIS[28][29] enables us to monitor the environment where the malware is executing and records the system call sequences from this process. The system call sequence that we retrieve shows the program executing behaviors chronologically.

3.3 System Call Frequency Encoding Method

This paper proposes a system call encoding method to transform system calls into interval behavior feature vectors and maintain their original features. After recording the system calls from VIS, we treat system calls as an unordered n-gram interval behavior feature vector and count the system calls occurrence frequency in a vector. After this method, we decrease the problem of complexity and the large size of data. The following sequence M represents the malware or benignware system call sequence, s indicate the system calls and N is the order of system calls.

$$M = \left[s_0, s_1, s_2, ..., s_N \right] \tag{1}$$

Because system call sequence is a huge amount which is approximate hundreds of thousands each malware in 2 minutes recording. These huge amount system calls are then split into n-gram interval behavior. Each interval behavior represents a kind of malware or benignware activity in a fixed time interval. As below, n indicates that how many system calls in an interval behavior. M comprise those interval behaviors $m_{N/n}$ and m_0 is the first interval behavior of this malware.

$$m_0 = \left[s_0, s_1, s_2, ..., s_n \right] \tag{2}$$

$$M = \left[m_0, m_1, m_2, ..., m_{N/n} \right] \tag{3}$$

There are hundreds of system calls in Linux, only 52 system calls are representative of the program behaviors[13]. L is a 52 attributes array which only contains the system

DOI:10.6814/THE.NCCU.MIS.025.2018.A05

call we use. The superscript s is a system call belongs to L and the subscript is the order of recorded system calls in an interval behavior m_0.

$$L = \left[l^0, l^1, l^2, ..., l^{51} \right] \tag{4}$$

$$s \in L \tag{5}$$

$$m_0 = \left[s_0^2, s_1^0, s_2^3, s_3^{10}, ..., s_n^l \right] \tag{6}$$

In order to transform interval behavior m_0 to interval behavior feature vector b_0, the system calls s in m_0 are counted and grouped by the element in L.

$$f(x) = count(s^x) \tag{7}$$

$$b_0 = \left[f(l^0), f(l^1), f(l^2), ..., f(l^{51}) \right] \tag{8}$$

The interval behavior feature vector b_0 is the transformed interval behavior m_0. A malware or benignware behavior M is constituted of an interval behavior feature vectors b_0.

$$M = \left[b_0, b_1, b_2, ..., b_{N/n} \right] \tag{9}$$

3.4 Unsupervised Learning Clustering

Unsupervised learning is a machine learning algorithm which infers the patterns of data without labels. The common tasks to use unsupervised learning algorithm are clustering, includes k-means, principal component analysis, autoencoders and hierarchical clustering. The unsupervised learning algorithm we use is the growing hierarchical self-organizing map (GHSOM) which belongs to hierarchical clustering. GHSOM is the self-organizing

DOI:10.6814/THE.NCCU.MIS.025.2018.A05

map based on the distribution of data. The GHSOM algorithm depends on the variance of the clusters in the same hierarchical layer to decide the size of the self-organizing map. If the variance is too big in the same layer, the map will grow the self-organizing map in a horizontal way to fit variance of data. GHSOM also grows the self-organizing map in a hierarchical way which based on variance within the cluster. As long as we set the target variance between clusters and within the cluster, GHSOM algorithm can keep tuning the growing hierarchical self-organizing map to fit the target variance between clusters and within clusters. The final map is a tree structure of the self-organizing map in Figure 3.

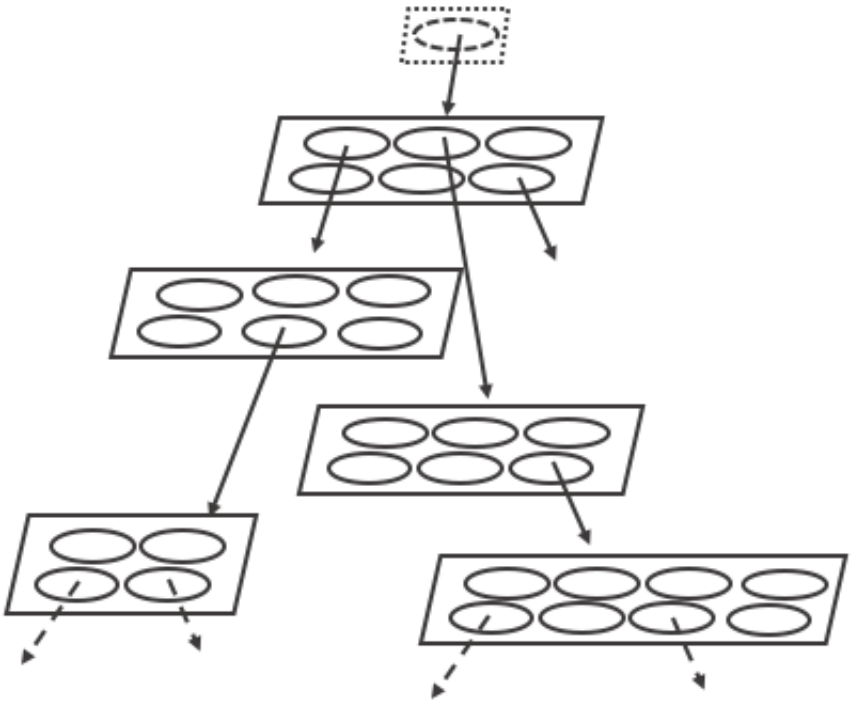

Figure 3: The structure of GHSOM

Unsupervised learning clustering analysis is deployed on iOS executable[30] and malware with SOM(Self-Organizing Map)[31] algorithm to identify the clusters of similar behaviors on programs. To obtain more accurate clustering analysis result, we utilize GHSOM algorithm, a growing multi-layers map which consists of independent SOMs in a hierarchical way. The interval behavior feature vectors of the program are forwarded as input data for GHSOM algorithm. GHSOM algorithm enables the system to classify the interval behavior feature vectors of malware into corresponding behavior clusters. Each

DOI:10.6814/THE.NCCU.MIS.025.2018.A05

behavior feature vector represents a type of malware behavior over a period of time and similar interval behavior feature vectors are classified into the same cluster by GHSOM algorithm. All interval behavior vectors are the input of GHSOM, so vectors are appended to a malware dataset. M_{all} is a malware dataset which consists of many M_i, malware interval behavior sequences. M_i contains the interval behavior vectors representing the order of malware behaviors $b_{N/n}$ in time sequence.

$$M_{all} = \{M_0, M_1, M_2, ..., M_i\} \tag{10}$$

$$i \in N \tag{11}$$

$$M_i = \left[b_0, b_1, b_2, ..., b_{N/n}\right] \tag{12}$$

After the processing of unsupervised learning clustering, GHSOM, each behavior feature vector b_i is classified into a behavior cluster c_i. Interval behavior feature vectors b_i are replaced with interval behavior clusters c_i to present the program behaviors.

$$M_i = \left[c_0, c_1, c_2, ..., c_{N/n}\right] \tag{13}$$

3.5 The Hierarchical SOM Encoding Method

Each behavior vector belongs to a cluster c_i in a self-organizing map on the GHSOM layer, in other words, a cluster now is the representative of a behavior feature vector. Since the clustering result is not machine-operatable for RNN, the transformation for interval behavior clusters c_i is needed. To transform GHSOM output to RNN input, this paper proposes three cluster encoding method which are decimal encoding method, weighted one-hot encoding method, and map projection embedding method[32].

DOI:10.6814/THE.NCCU.MIS.025.2018.A05

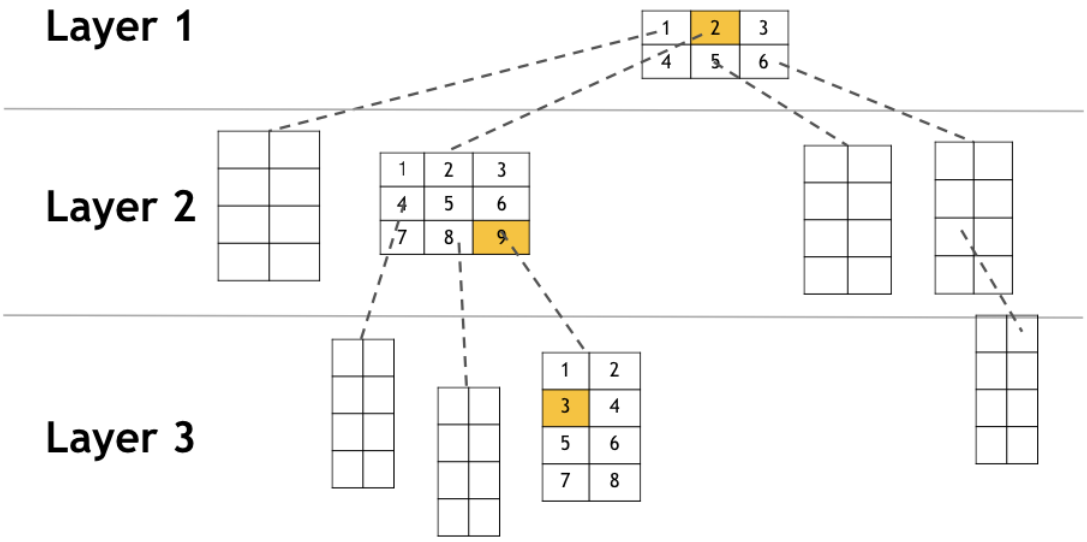

Figure 4: Decimal encoding method

3.5.1 Decimal Encoding Method

Due to the hierarchical architecture of GHSOM, it is a three-dimensional distribution clustering which is difficult to find a way to represent the result without compressing the data. Decimal digits are ten times between each other digits which provide a similar hierarchical structure so the processing flow is processed by the layers of GHSOM results step by step. As shown in Figure 4, the GHSOM consists of the top layer containing only one self-organizing map with clusters, the middle layer and the bottom layer containing several maps. Each cluster in a self-organizing map is numbered in an numeral order. A vector is only classified to the final state of the cluster, in other words, the cluster without the child layer. For example, in Figure 4, the number with the yellow background in each layer is the cluster where a vector belongs. We add all decimal numbers in three layers to one decimal number '293' as a cluster representative. If a top or middle layer doesn't contain child layer, we append zero to the end of the decimal numeral. In order to retrieve an accurate result, after all, representative of clusters is generated, the decimal numeral is multiplied by 0.001.

DOI:10.6814/THE.NCCU.MIS.025.2018.A05

$$l_1 = 4 \tag{14}$$

$$l_2 = 2 \tag{15}$$

$$l_3 = 3 \tag{16}$$

$$c_i = l_1 l_2 l_3 * 0.001 \tag{17}$$

$$c_i = 0.423 \tag{18}$$

3.5.2 Weighted One-hot Encoding Method

The weighted one-hot encoding method is derived from the decimal encoding method using the classification feature of one-hot encoding method to label different layer and cluster position. We use the weighted one-hot encoding method to convert interval behavior clusters into interval behavior clusters vectors. The interval behavior clusters vector consists of several one-hot vectors corresponding to the number of hierarchical SOM layers. As shown in Figure 4, several self-organizing maps are located in each layer. The SOM with most clusters is the one-hot vector benchmark of the current layer, for example, a map with n cluster is a n length vector. If the cluster is classified to the map, the vector of the specific index which corresponding to the order of the map values 1 as a one-hot cluster vector. If the cluster is not in the layer, the vector will append one binary bit as a representative of the nonexistent cluster. Then, we modify the one-hot cluster vectors with the weighted function because the one-hot vectors are losing the hierarchical feature of GHSOM. To recover the hierarchical feature we set the value 1 of the first vector to 4, second vector set to 2 and third vector set to 1. Based on *cosine similarity*, a measurement of the similarity between vectors, the similarity of weighted one-hot vectors is higher if the vectors are in the same layer. In Figure 4, there are 6 clusters in the first GHSOM layer and interval behavior cluster vector l_1 in the second cluster is encoded to 040000 in one-hot encoding method. The second layer contains 9 clusters in the GHSOM layer, and interval behavior cluster vector l_2 in the last cluster of the SOM is encoded

DOI:10.6814/THE.NCCU.MIS.025.2018.A05

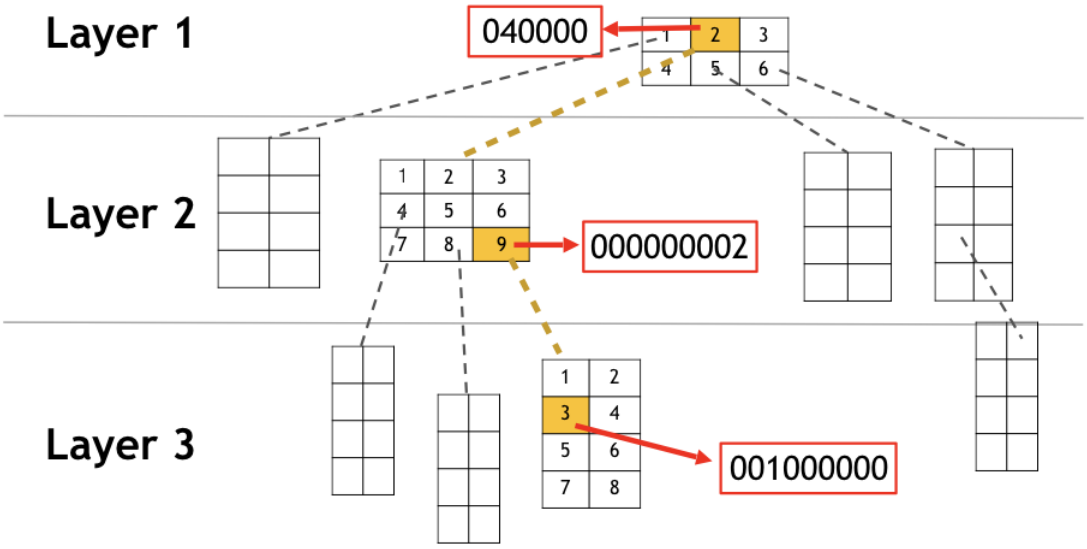

Figure 5: The weighted one-hot encoding method

to 000000002 in one-hot encoding. The third layer contains 8 clusters in the GHSOM layer, and interval behavior cluster vector l_3 in the third cluster is encoded as 00100000 in one-hot encoding. Sometimes the interval behavior cluster vector l_3 is not classified into the third layer, so we append a binary bit to the end of the encoding cluster vector as the representative. In this case, cluster vector 001000000 appends a place for the interval behavior cluster not classified into the third layer. Finally, we join all l_i together, and it ends with [040000000000002001000000] to represent a complete 24 bits interval behavior cluster vector. l_i is the one-hot encoding result in i layer and c_i is the corresponding cluster of GHSOM result. We show the binary encoding result map of the GHSOM with the hierarchical SOM binary encoding method.

$$l_1 = 040000 \tag{19}$$

$$l_2 = 000000002 \tag{20}$$

$$l_3 = 001000000 \tag{21}$$

$$c_i = \left[l_1, l_2, l_3 \right] \tag{22}$$

DOI:10.6814/THE.NCCU.MIS.025.2018.A05

3.6 Malware Family Labeling

We use the labels defined by different antivirus software vendors from VirusTotal[33]. Due to the lack of naming standard of malware family, we use the "AVClass: A Tool for Massive Malware Labeling"[34], which can label the most suitable family name for each malware. First, AVClass lists all the labels made by antivirus software companies. Then it removes the duplicate malware labels, removes the suffix characters, and tokenize each character to mark it. Finally, AVClass leaves the most representative tokens and select the label with the highest number of recurrences as a family label of the malware. The paper indicates that the accuracy of the AVClass tool clustering can be as high as 0.939, and it will fluctuate with the data set.

3.7 Recurrent Neural Network

Since the input data is time sequential data, the recurrent neural network outperforms many related statistical models, such as hidden Markov models[35]. Therefore, this paper uses the recurrent neural network to analyze the sequential interval behavior vectors. We use Long-term memory model to analyze the sequential interval behavior vectors to judge whether the program is malicious [36] and determine the family malware belongs to. Because the length of the behavior vectors of the system call is extended, the general recursive neural network will cause the gradient vanishing problem[14], so long short-term memory with forgetting gate is chosen[17]. In order to facilitate the input to the recursive neural network, we use the hierarchical SOM binary encoding method in the previous section to fit the LSTM requirement. We append all complete interval behavior cluster vectors of a malware to make malware behaviors dataset in a two-dimensional matrix M_{all}. M_i is fixed length, and comprise interval behavior cluster vectors c_0^i, the superscript and subscript of which is the malware order in M_{all} and the order of interval behavior cluster vectors respectively. The subscript N/n in $c_{N/n}^i$ represent the N system call numbers and n-gram in the previous section.

DOI:10.6814/THE.NCCU.MIS.025.2018.A05

$$M_{all} = \begin{bmatrix} M_0 = c_0^0, c_1^0, c_2^0, ..., c_{N/n}^0 \\ M_1 = c_0^1, c_1^1, c_2^1, ..., c_{N/n}^1 \\ M_2 = c_0^2, c_1^2, c_2^2, ..., c_{N/n}^2 \\ ... \\ M_i = c_0^i, c_1^i, c_2^i, ..., c_{N/n}^i \end{bmatrix} \tag{23}$$

We use the interval behavior cluster vectors of all malware M_{all} as the input data for Long short-term memory model. 10-fold cross validation is also adopted where 90% of the random samples are training data, and 10% are test data. In order to have the better results of the long short-term memory model, we train the model with batch size and uses the many-to-one pattern in long short-term memory model with softmax layer, as shown in Figure 5. After all elements in the dataset matrix M_{all} are forwarded to long short-term memory model, the softmax layer is used. The softmax layer is used to predicts the family classification label by calculating the family probability of this malware and family with the highest probability is chosen. The accuracy of the LSTM model can be acquired by comparing the prediction family with the ground truth family.

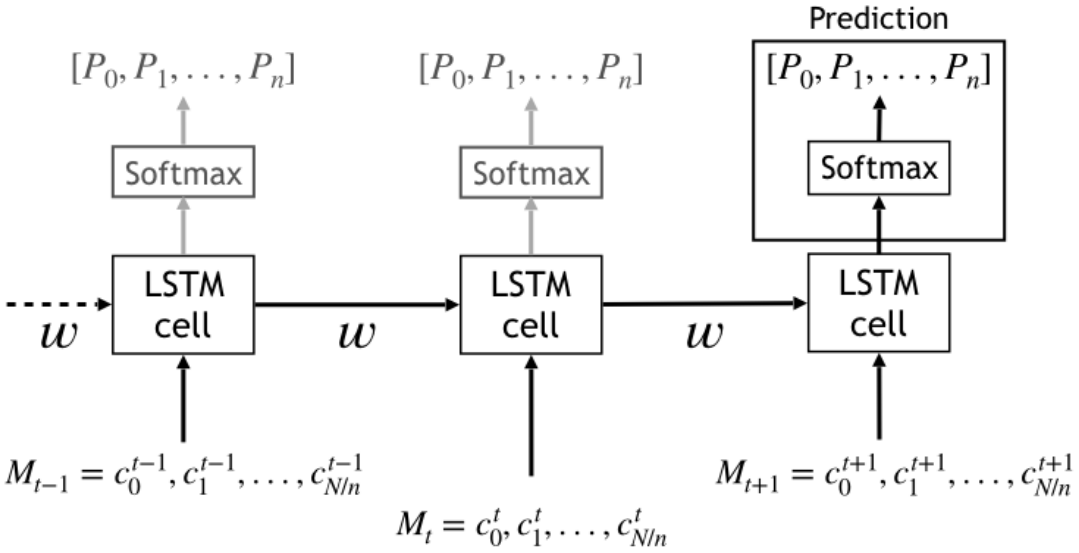

Figure 6: Many-to-one LSTM model with softmax layer

DOI:10.6814/THE.NCCU.MIS.025.2018.A05

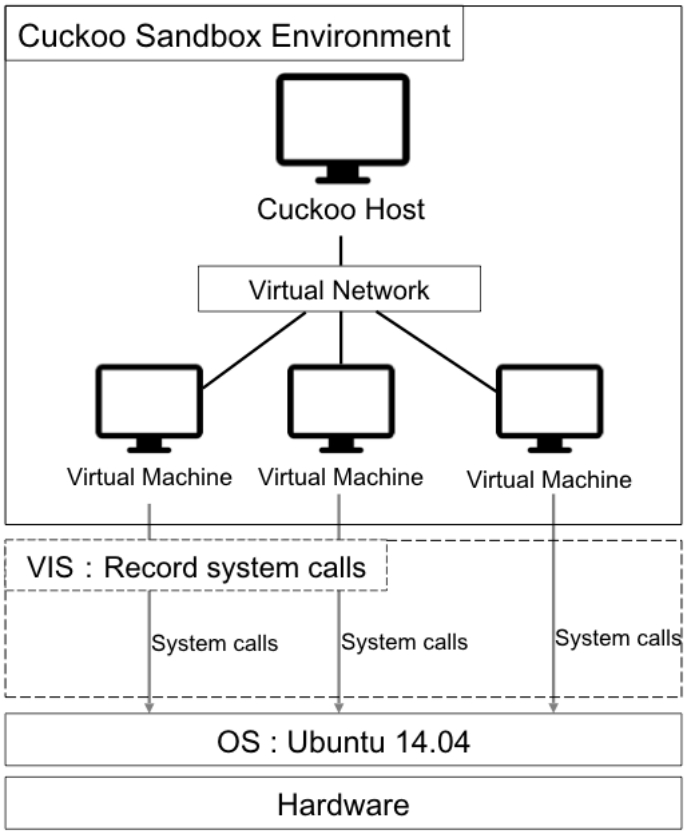

Figure 7: System architecture

4 Experiment

In this section, we demonstrate the experimental process and the results. The following is the implementation of the growing hierarchical self-organizing map algorithm and recurrent neural network for malware detection and classification.

4.1 System Architecture

The system consists of three parts: malware behavior monitor, behavior clustering with GHSOM, and sequential analysis with RNN. We monitor the invoked system calls of VM at malware execution. To provide an appropriate virtual environment, CPU we use is Intel(R) Xeon(R) E5630 2.53GHz, with a total memory of 96GB. The operating system is Ubuntu 14.04 and the programs executed in a virtual machine, cuckoo sandbox with Window 7 operating system inside. While the program execution, we use VIS and *strace* [37] to trace the system calls invoked by Cuckoo Sandbox. The strace is a system call

DOI:10.6814/THE.NCCU.MIS.025.2018.A05

tracing tool for Linux. We trace and save the system calls requested from the host. The recorded system calls constitute the behavior pattern of the program in chronological order. Due to the verbosity of system calls, we simplify the behavior with the n-gram method. Each n-gram item is the behavior of the program in a fixed period of time. Based on the sequence of interval behavior in chronological order, the detection and classification of malware are available. After tracing and pre-processing system calls sequences, the sequences are fed to GHSOM algorithm. In GHSOM alogrithm, we set tau1 and tau2 to 0.1 and 0.01 respectively to generate a reasonable size map. After clustering, each interval behavior is classified and corresponds to an interval behavior cluster. Then, the hierarchical SOM binary encoding method is used to indicate the cluster of interval behavior in GHSOM. The input data of the recurrent neural network is the interval behavior sequences. The output of the recurrent neural network is a one-hot array indicating the predicted malware family. We tune the hyperparameters to get the best training results and use LSTM to obtain the predicted malware family label comparing with the ground truth label. Because tensorflow performs better on GPU than CPU, we use the NVIDIA GTX 1080 GPU to mark the malware families and detect the malware.

4.2 Experiment Dataset

In this paper, we selected 5,000 malware and 2,851 benign software for Windows XP from National Taiwan University database[38]. We use VIS to directly record the system calls of the entire virtual machine. Strace traces the system call while the virtual machine is running, and sets the recording time to 3 minutes. Tracing system calls of the virtual machine is interrupted when the time is up, and the monitoring is also terminated. According to Chiu[13], we selected 52 type of system calls which has the significant effect on malware behavior analysis as attributes from all types of system calls. The attributes, such as read, ppoll, futex etc., are shown in Table 1. The recorded system call is set as an n-gram behavior activity, and sum up the number of different types of system calls

DOI:10.6814/THE.NCCU.MIS.025.2018.A05

to generate a malware behavior vector. From 500,000 to 3,500,000 system calls can be recorded in 3 minutes, that is, 50 to 3500 interval behavior vectors. Due to the different characteristics of malware, the number of malware behavior vectors are more or less.

Table 1: System calls reference [2]

System call	Name
ppoll	wait for some event on a file descriptor
tgkill	send a signal to a thread
read	read from a file descriptor
ioctl	control device
exit	terminate the calling process
close	close a file descriptor
brk	change data segment size
madvice	give advice about use of memory
open	open and possibly create a file
futex	fast user-space locking

4.3 Unsupervised Learning Clustering

In unsupervised learning clustering, we forward all interval behavior vectors of the program into the growing hierarchical self-organizing map clustering algorithm. After setting the parameters tau1 and tau2 to implement GHSOM algorithm and generating the growing hierarchical self-organizing map, we vectorize the clustering with the hierarchical SOM binary encoding method so that it can be input to the recurrent neural network. We obtain the position of each behavior in different layers and all the cluster states of the layer. The 1000-gram interval behavior input for GHSOM is shown in Figure 8.

```
52
rt, tgkill, read, ppoll, futex, write, ioctl, exit, close, SIGTERM, poll, munmap, exited, fcntl, mmap, accept4, brk, recvmsg, fstat, clone, recvfrom, sendto,
mprotect, getsockname, getpeername, lseek, set, uname, connect, prctl, getgid, open, fstatfs, access, getdents, signalfd, getrlimit, listen, stat, setsockopt,
 openat, readlink, unlink, gettid, getuid, socket, madvise, bind, eventfd2, statfs, getegid, geteuidcompute
0ab6c8a6c07925618c58be1751bfecb6544429c8a17ea2d773c799f4ce98a0c3_33, 0, 0, 333, 333, 333, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0ab6c8a6c07925618c58be1751bfecb6544429c8a17ea2d773c799f4ce98a0c3_34, 0, 0, 333, 333, 333, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0ab6c8a6c07925618c58be1751bfecb6544429c8a17ea2d773c799f4ce98a0c3_35, 0, 0, 333, 333, 333, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
```

Figure 8: 1000-gram interval behavior vectors

DOI:10.6814/THE.NCCU.MIS.025.2018.A05

4.4 Recurrent Neural Network

After two implements of the hierarchical SOM binary encoding method, each malware has a sequence of corresponding malware behavior cluster vectors, as shown in Table 2 and Table 3, so we feed them into the recurrent neural network to generate a malware family probability vector. In order to compare the performance under various conditions, we train recurrent neural networks with different parameters. This many-to-one input model produces an output, which is the predicted family name of the malware. In this experiment, we used 10-fold cross-validation to confirm the accuracy of the experiment. The data is divided into ten equal segments. In each test, we select nine parts as training data, and the remaining is used as test data. After collecting all the output data, the test results are averaged ten times, and the accuracy of the experiment is obtained.

Table 2: Interval behavior cluster vectors by weighted one-hot encoding method

$$M_i = \begin{bmatrix}
0. & 4. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 2. & 0. & 0. & 1. & 0. & 0. & 0. & 0. \\
0. & 4. & 0. & 0. & 0. & 0. & 2. & 0. & 0. & 0. & 0. & 0. & 1. & 0. & 0. & 0. & 0. \\
0. & 4. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 2. & 0. & 0. & 0. & 0. & 1. & 0. & 0. \\
0. & 0. & 4. & 0. & 0. & 2. & 0. & 0. & 0. & 0. & 1. & 0. & 0. & 0. & 0. & 0. & 0. \\
0. & 4. & 0. & 0. & 0. & 0. & 2. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 1. \\
0. & 0. & 4. & 0. & 2. & 0. & 0. & 0. & 0. & 0. & 1. & 0. & 0. & 0. & 0. & 0. & 0. \\
0. & 4. & 0. & 0. & 0. & 0. & 0. & 2. & 0. & 0. & 1. & 0. & 0. & 0. & 0. & 0. & 0. \\
 & & & & & & & & & ... & & & & & & & \\
0. & 4. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 2. & 0. & 1. & 0. & 0. & 0. & 0. & 0. \\
4. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 2. & 0. & 0. & 1. & 0. & 0. & 0. & 0. & 0.
\end{bmatrix}$$

4.5 Malware Detection Experiment

In the malware detection experiment, we input the behavior vectors of malicious software and benign software into the recurrent neural network and determine whether the software is malicious by analyzing the program behavior. Table 4 represents the training and testing results with the different hyperparameters. From the testing results, we see that the batch size determines whether the model has a good result. When the batch is equal to 10, the probability of prediction whether the software is malicious is only 0.625, and the batch is raised to 20 the probability of prediction is increased to 0.859. The highest

DOI:10.6814/THE.NCCU.MIS.025.2018.A05

Table 3: Interval behavior cluster vectors by decimal encoding method

$$M_i = \begin{matrix} 0.260 \\ 0.260 \\ 0.430 \\ 0.430 \\ 0.316 \\ 0.334 \\ 0.314 \\ 0.314 \\ 0.315 \\ ... \\ 0.312 \\ 0.313 \\ 0.312 \\ 0.240 \\ 0.140 \end{matrix}$$

Table 4: Result of malware detection experiment

Epoch	Batch size	Testing accuracy
600	10	0.625
600	20	0.859
600	40	0.938
600	100	0.980
600	200	0.861
300	10	0.625
300	20	0.625
300	40	0.625
300	100	0.892
300	200	0.914

accuracy occurs when the batch is equal to 100. At this time, the accuracy is 0.98. When we continue to increase the batch to 200, the accuracy slightly decreases to 0.961. This should be an overfitting situation. The best batch size should be 100.

4.6 Malware Family Classification Experiment

We use 18 categories generated from AVClass tool ,and take these 18 categories as our reference benchmarks to judge the accuracy of the training model. In this experiment, we set interval behavior vectors as 1000-gram and 2000-gram and compare the effect between two different n-gram vector. While calculating the testing accuracy, we consider

DOI:10.6814/THE.NCCU.MIS.025.2018.A05

the top two high possibilities of the guess as top 2 accuracies because sometimes the same behavior may be traced from different malware families.

Table 5: Training & testing accuracy of 1000-gram vectors by decimal encoding method

Epoch \ Portion	100%	50%	25%	10%	5%	Iteration
100	0.57	0.58	0.54	0.55	0.48	0.41
200	0.61	0.64	0.5	0.44	0.59	0.37
300	0.65	0.63	0.53	0.55	0.68	0.42
400	0.71	0.68	0.47	0.6	0.76	0.40
500	0.81	0.69	0.56	0.6	0.72	0.36
600	0.81	0.74	0.74	0.71	0.74	0.39
700	0.87	0.69	0.77	0.63	0.73	0.17
800	0.81	0.84	0.71	0.75	0.68	0.36
900	0.86	0.77	0.70	0.76	0.72	0.31
1000	0.9	0.81	0.79	0.66	0.72	0.36
Testing Accuracy	0.665	0.693	0.664	0.677	0.719	0.342
Top 2 Testing Accuracy	0.771	0.820	0.821	0.807	0.867	0.5

Table 6: Training & testing accuracy of 1000-gram vectors by weighted one-hot encoding method

Epoch \ Portion	100%	50%	25%	10%	5%	Iteration
100	0.25	0.32	0.45	0.41	0.33	0.24
200	0.47	0.59	0.60	0.57	0.53	0.21
300	0.52	0.64	0.75	0.75	0.72	0.2
400	0.58	0.64	0.82	0.85	0.85	0.19
500	0.65	0.67	0.91	0.88	0.87	0.22
600	0.74	0.77	0.83	0.93	0.93	0.26
700	0.75	0.75	0.91	0.90	0.96	0.28
800	0.81	0.82	0.88	0.94	0.87	0.25
900	0.88	0.84	0.83	0.90	0.91	0.32
1000	0.9	0.79	0.91	0.93	0.82	0.30
Testing Accuracy	0.152	0.141	0.172	0.09	0.113	0.144
Top 2 Testing Accuracy	0.252	0.250	0.2781	0.226	0.222	0.264

In Table 5 and Table 6, 1000-gram vectors are transformed by decimal encoding and weighted one-hot method respectively and then forwarded into RNN model. Each table is trained in 1000 epochs and the RNN input are divided in several portion, such as 100% , 50%, 25%, 10%, 5%, iteration, to retrieve the best accuracy of training model. In Table 5, the highest testing accuracy and the top 2 testing accuracy occurs at 5% portion of

DOI:10.6814/THE.NCCU.MIS.025.2018.A05

RNN input. As data shown, 100% may encounter the overfitting problem. In Table 6, the highest testing accuracy and the top 2 testing accuracy occurs at iteration portion of RNN input. As the result shown, the training accuracy are better than Table 5 but the testing accuracy are much lower which may be the overfitting problem due to the weighted one-hot encoding method.

In Table 7 and Table 8, 2000-gram vectors are transformed by decimal encoding and the weighted one-hot method respectively and then forwarded into the RNN model. Each table is trained in 500 epochs and the RNN input are divided into several portions, such as 100% , 50%, 25%, 10%, 5%, iteration, to retrieve the best accuracy of training model. In Table 7, the highest testing accuracy and the top 2 testing accuracy occurs at 25% portion of RNN input. In Table 8, the testing accuracy and the top 2 testing accuracy are not significantly different. As the result shown, the weighted one-hot encoding method performs the worse result. Figure 9,10,11 and 12, has shown the line chart of the testing accuracy.

Table 7: Training & testing accuracy of 2000-gram vectors by decimal encoding method

Portion / Epoch	100%	50%	25%	10%	5%	Iteration
100	0.73	0.67	0.67	0.5	0.45	0.39
200	0.76	0.78	0.7	0.56	0.66	0.42
300	0.74	0.77	0.65	0.66	0.69	0.27
400	0.74	0.75	0.8	0.68	0.72	0.32
500	0.83	0.87	0.81	0.63	0.68	0.43
Testing Accuracy	0.731	0.710	0.751	0.690	0.657	0.401
Top 2 Testing Accuracy	0.876	0.840	0.890	0.845	0.844	0.496

5 Conclusion

This paper proposes a new malware detection and classification framework. We use the growing hierarchical self-organizing map to group program behaviors, obtain a record of all kinds of program operations on virtual machines, and use a recurrent neural network to perform the behavior sequence analysis. The experimental results of this paper in the

DOI:10.6814/THE.NCCU.MIS.025.2018.A05

Table 8: Training & testing accuracy of 2000-gram vectors by weighted one-hot encoding method

Portion / Epoch	100%	50%	25%	10%	5%	Iteration
100	0.34	0.26	0.28	0.28	0.21	0.17
200	0.45	0.46	0.51	0.42	0.32	0.15
300	0.48	0.58	0.62	0.56	0.47	0.22
400	0.45	0.6	0.71	0.66	0.57	0.21
500	0.62	0.68	0.67	0.71	0.65	0.2
Testing Accuracy	0.144	0.137	0.166	0.161	0.109	0.19
Top 2 Testing Accuracy	0.310	0.289	0.268	0.267	0.258	0.33

malware detection, when the batch size is smaller, the accuracy is not expected to be good, only 0.625 accuracy, and after adjusting the batch size, it can reach 0.98 accuracy, we think that the batch size for the prediction model has a significant impact. The classification accuracy reaches 0.719 at testing accuracy and 0.867 at top 2 testing accuracy when 1000-gram vectors are encoded by the decimal encoding method. In the future, we will delve into how to deal with data noise problems and how to make classification labels more accurate, as well as extend the flexibility of the model for data so that we can cope with a larger and more complex malware behavior analysis in the future.

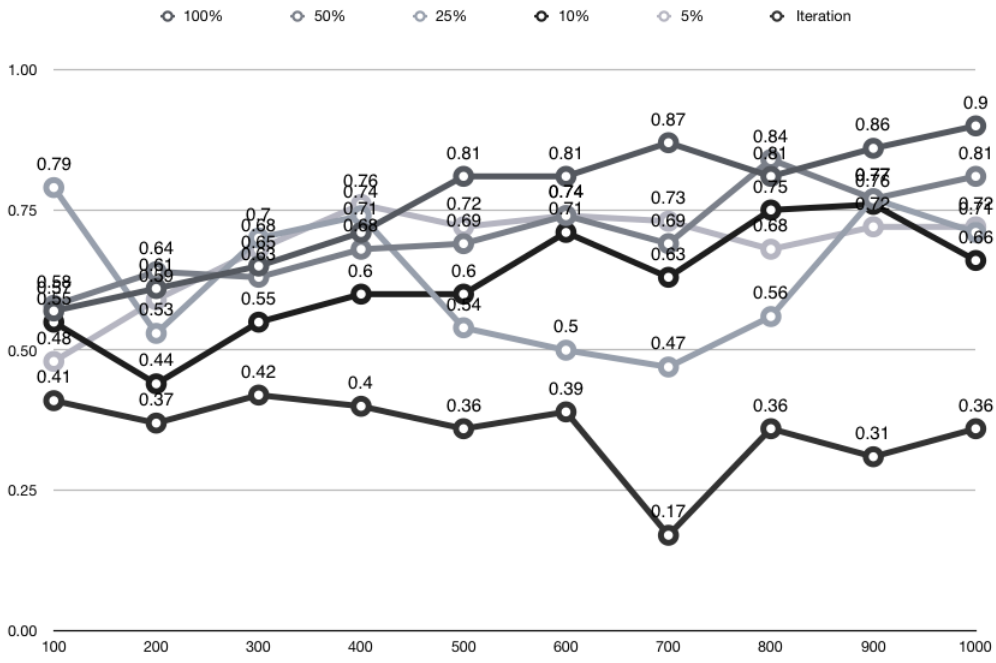

Figure 9: Testing accuracy of 1000-gram decimal encoding vector

DOI:10.6814/THE.NCCU.MIS.025.2018.A05

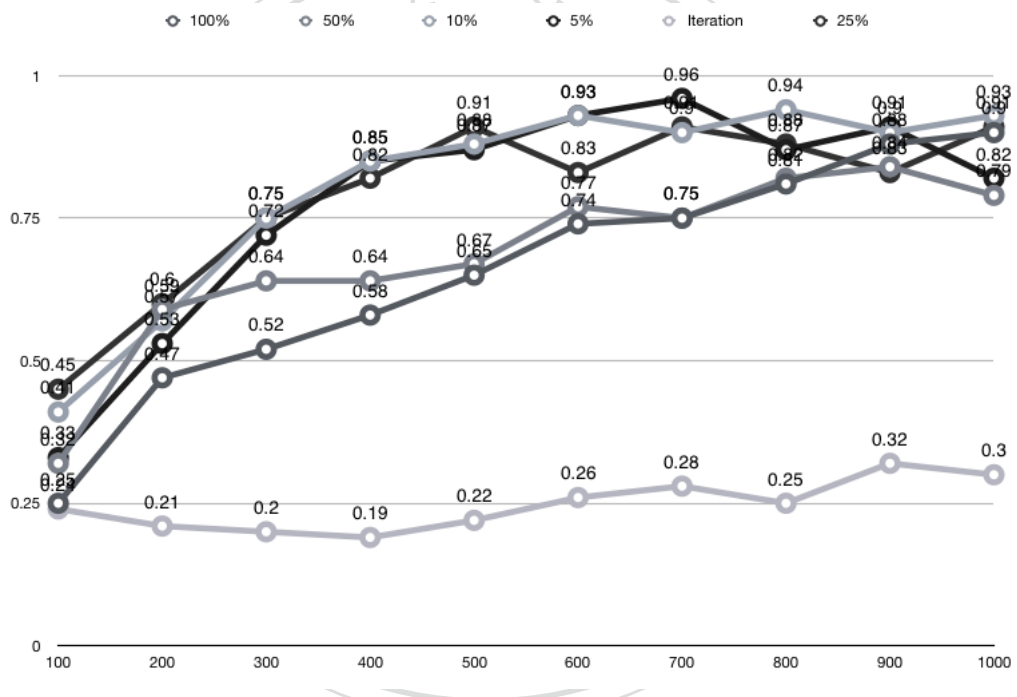

Figure 10: Testing accuracy of 1000-gram weighted one-hot encoding vector

DOI:10.6814/THE.NCCU.MIS.025.2018.A05

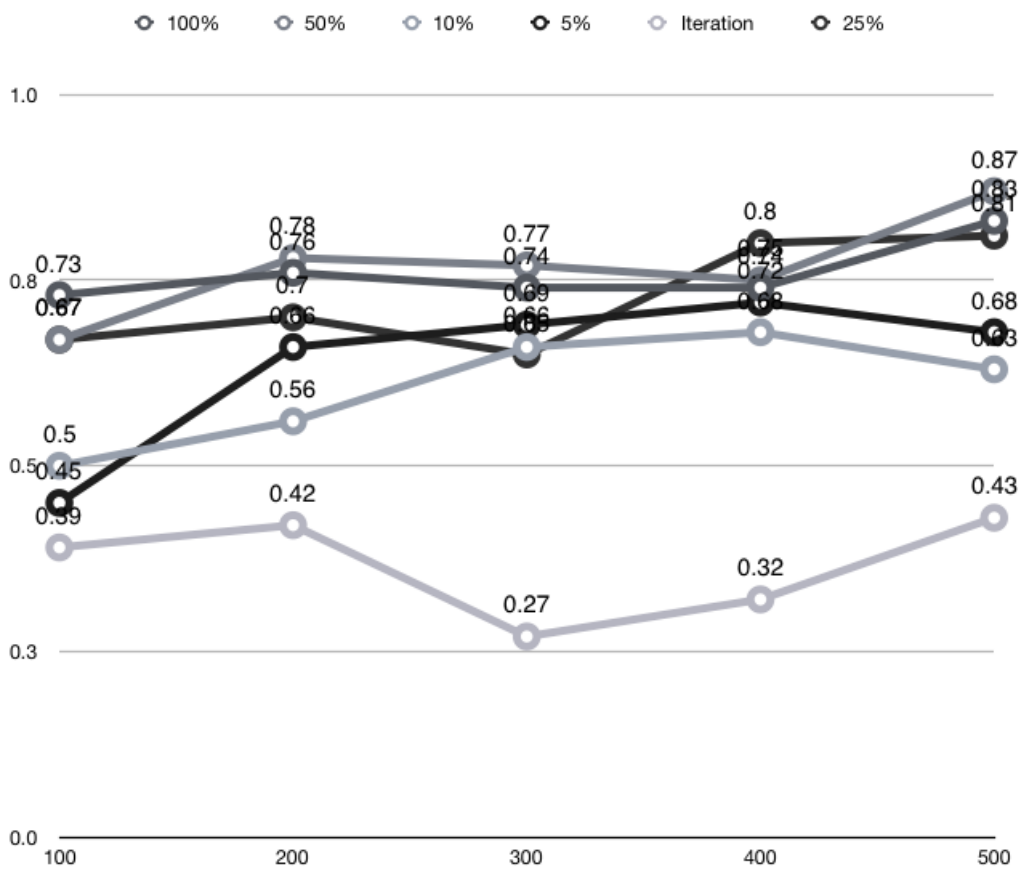

Figure 11: Testing accuracy of 2000-gram decimal encoding vector

DOI:10.6814/THE.NCCU.MIS.025.2018.A05

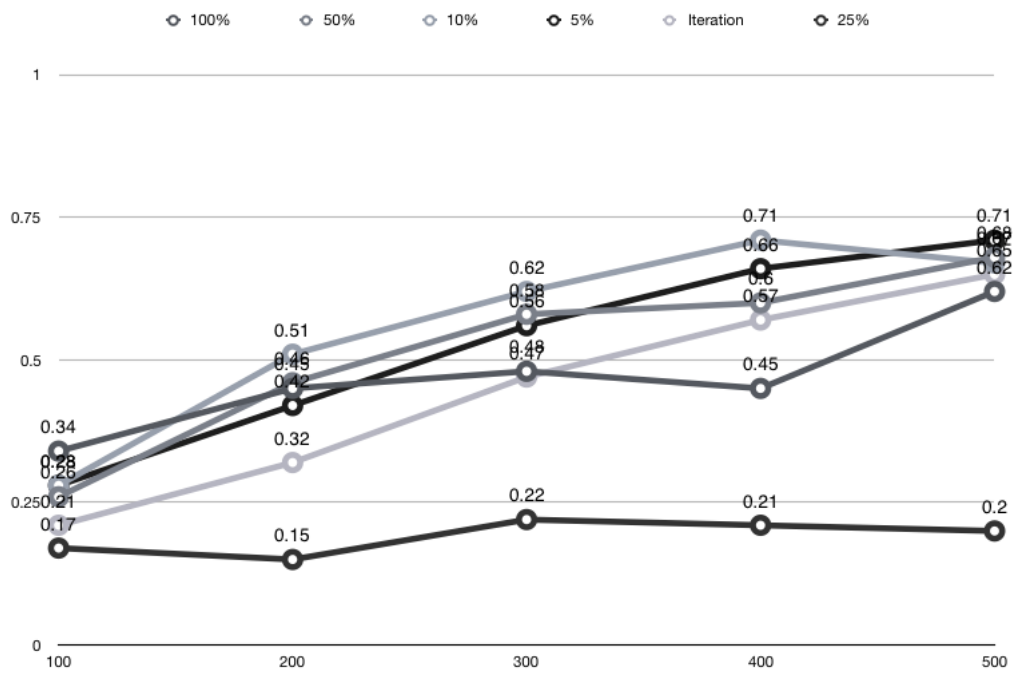

Figure 12: Testing accuracy of 2000-gram weighted one-hot encoding vector

DOI:10.6814/THE.NCCU.MIS.025.2018.A05

References

[1] A.-r. M. https://commons.wikimedia.org/wiki/User:BiObserve (Raster version previously uploaded to Wikimedia)Alex Graves and G. H. (original)Eddie Antonio Santos (SVG version with TeX math), "Peephole long short-term memory," "[CC BY-SA 4.0 (https://creativecommons.org/licenses/by-sa/4.0)], via Wikimedia Commons".

[2] "Linux syscall reference," https://syscalls.kernelgrok.com/, [Online; accessed 11-August-2018].

[3] R. J. Canzanese Jr, "Detection and classification of malicious processes using system call analysis," Ph.D. dissertation, Drexel University, 2015.

[4] T. Moore, D. J. Pym, C. Ioannidis *et al.*, *Economics of information security and privacy*. Springer, 2010.

[5] N. Idika and A. P. Mathur, "A survey of malware detection techniques," *Purdue University*, vol. 48, 2007.

[6] "Manalyze," https://github.com/JusticeRage/Manalyze, [Online; accessed 4-May-2018].

[7] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff, "A sense of self for unix processes," in *Security and Privacy, 1996. Proceedings., 1996 IEEE Symposium on*. IEEE, 1996, pp. 120–128.

[8] M. Rhode, P. Burnap, and K. Jones, "Early stage malware prediction using recurrent neural networks," *arXiv preprint arXiv:1708.03513*, 2017.

[9] X. Wang and S. M. Yiu, "A multi-task learning model for malware classification with useful file access pattern from api call sequence," *arXiv preprint arXiv:1610.05945*, 2016.

DOI:10.6814/THE.NCCU.MIS.025.2018.A05

[10] B. Kolosnjaji, A. Zarras, G. Webster, and C. Eckert, "Deep learning for classification of malware system call sequences," in *Australasian Joint Conference on Artificial Intelligence*. Springer, 2016, pp. 137–149.

[11] S. Tobiyama, Y. Yamaguchi, H. Shimada, T. Ikuse, and T. Yagi, "Malware detection with deep neural network using process behavior," in *Computer Software and Applications Conference (COMPSAC), 2016 IEEE 40th Annual*, vol. 2. IEEE, 2016, pp. 577–582.

[12] R. Pascanu, J. W. Stokes, H. Sanossian, M. Marinescu, and A. Thomas, "Malware classification with recurrent networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*. IEEE, 2015, pp. 1916–1920.

[13] C.-H. Chiu, J.-J. Chen, and F. Yu, "An effective distributed ghsom algorithm for unsupervised clustering on big data," in *Big Data (BigData Congress), 2017 IEEE International Congress on*. IEEE, 2017, pp. 297–304.

[14] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997. [Online]. Available: http://dx.doi.org/10.1162/neco.1997.9.8.1735

[15] H. Sak, A. Senior, and F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling," in *Fifteenth annual conference of the international speech communication association*, 2014.

[16] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[17] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," 1999.

DOI:10.6814/THE.NCCU.MIS.025.2018.A05

[18] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[19] T. Mikolov, M. Karafiát, L. Burget, J. Černockỳ, and S. Khudanpur, "Recurrent neural network based language model," in *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

[20] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*. IEEE, 2013, pp. 6645–6649.

[21] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[22] T. Kohonen, "The self-organizing map," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1464–1480, 1990.

[23] A. Rauber, D. Merkl, and M. Dittenbach, "The growing hierarchical self-organizing map: exploratory analysis of high-dimensional data," *IEEE Transactions on Neural Networks*, vol. 13, no. 6, pp. 1331–1341, 2002.

[24] H. Shi, T. Hamagami, K. Yoshioka, H. Xu, K. Tobe, and S. Goto, "Structural classification and similarity measurement of malware," *IEEJ Transactions on Electrical and Electronic Engineering*, vol. 9, no. 6, pp. 621–632, 2014.

[25] W. Shuwei, W. Baosheng, Y. Tang, and Y. Bo, "Malware clustering based on snn density using system calls," in *International Conference on Cloud Computing and Security*. Springer, 2015, pp. 181–191.

[26] M. Dittenbach, D. Merkl, and A. Rauber, "The growing hierarchical self-organizing map," in *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, vol. 6. IEEE, 2000, pp. 15–19.

DOI:10.6814/THE.NCCU.MIS.025.2018.A05

[27] C. Guarnieri, A. Tanasi, J. Bremer, and M. Schloesser, "The cuckoo sandbox," 2012.

[28] Y.-H. Li, Y.-R. Tzeng, and F. Yu, "Viso: Characterizing malicious behaviors of virtual machines with unsupervised clustering," in *Cloud Computing Technology and Science (CloudCom), 2015 IEEE 7th International Conference on.* IEEE, 2015, pp. 34–41.

[29] S.-W. Lee and F. Yu, "Securing kvm-based cloud systems via virtualization introspection," in *System Sciences (HICSS), 2014 47th Hawaii International Conference on.* IEEE, 2014, pp. 5028–5037.

[30] F. Yu, S.-y. Huang, L.-c. Chiou, and R.-h. Tsaih, "Clustering ios executable using self-organizing maps," in *Neural Networks (IJCNN), The 2013 International Joint Conference on.* IEEE, 2013, pp. 1–8.

[31] R.-S. Pirscoveanu, M. Stevanovic, and J. M. Pedersen, "Clustering analysis of malware behavior using self organizing map," in *Cyber Situational Awareness, Data Analytics And Assessment (CyberSA), 2016 International Conference On.* IEEE, 2016, pp. 1–6.

[32] S. Marinai, E. Marino, and G. Soda, "Embedded map projection for dimensionality reduction-based similarity search," in *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR).* Springer, 2008, pp. 582–591.

[33] "Virustotal," https://www.virustotal.com/en/, [Online; accessed 4-April-2018].

[34] M. Sebastián, R. Rivera, P. Kotzias, and J. Caballero, "Avclass: A tool for massive malware labeling," in *International Symposium on Research in Attacks, Intrusions, and Defenses.* Springer, 2016, pp. 230–253.

[35] Z. C. Lipton, J. Berkowitz, and C. Elkan, "A critical review of recurrent neural networks for sequence learning," *arXiv preprint arXiv:1506.00019*, 2015.

DOI:10.6814/THE.NCCU.MIS.025.2018.A05

[36] W. Hu and Y. Tan, "Black-box attacks against rnn based malware detection algorithms," *arXiv preprint arXiv:1705.08131*, 2017.

[37] "strace(1) - linux man page," https://linux.die.net/man/1/strace, [Online; accessed 5-April-2018].

[38] S.-W. Hsiao, Y.-N. Chen, Y. S. Sun, and M. C. Chen, "A cooperative botnet profiling and detection in virtualized environment," in *Communications and Network Security (CNS), 2013 IEEE Conference on*. IEEE, 2013, pp. 154–162.

 DOI:10.6814/THE.NCCU.MIS.025.2018.A05