JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 38, 1317-1334 (2022)
DOI: 10.6688/JISE.202211_38(6).0012

Learning Dynamic Malware Representation From Common Behavior

YI-TING HUANG[1,+], TING-YI CHEN[2], SHUN-WEN HSIAO[3], AND YEALI S. SUN[2]
[1]*Research Center for Information Technology Innovation*
Academia Sinica
Taipei, 115 Taiwan
[2]*Information Management*
National Taiwan University
Taipei, 106 Taiwan
[3]*Management Information Systems*
National Chengchi University
Taipei, 116 Taiwan
E-mail: ythuang@iis.sinica.edu.tw[+]; r06725035@g.ntu.edu.tw;
hsiaom@nccu.edu.tw; sunny@ntu.edu.tw

Malware analysis has been extensively investigated as the number and types of malware has increased dramatically. However, most previous studies use end-to-end systems to detect whether a sample is malicious, or to identify its malware family. In this paper, we introduce a framework composed of two components, RasMMA and RasNN, accounting for common characteristics within a family. While RasMMA extracts the common behaviors of malware, RasNN is designed to pretrain a composition of the common behaviors as malware representation. Different from the end-to-end models, the pre-trained malware representation can be fine-tuned with one additional output layer to apply other malware applications, such as family classification. We conduct broad experiments to determine the influence of individual framework components and the feasibility of a task-specific extension model. The results show that the proposed framework outperforms the other baselines, and also demonstrates that learned malware representation can be applied to other cybersecurity application and outperform the existing system.

Keywords: deep learning, dynamic analysis, malware behavior analysis, malware family classification, malware representation

1. INTRODUCTION

Malware, such as computer viruses, Internet worms, and Trojan horses, is a major threat in computer security, because malignant actors disrupt network services, destroy software or data, steal sensitive information, or take control of a host, often resulting in dramatic personal or business losses. As the number of malware has increased dramatically, it poses an even more significant threat to the Internet worldwide. Most malware programs are borrowed, modified, or combined from existing malware. In addition sophisticated obfuscation techniques, such as polymorphic engine, compression, encryp-

Received September 25, 2021; revised November 4, 2021; accepted December 4, 2021.
Communicated by Ying-Dar Lin.

tion, or packing tools [1,2] are applied to evade from detection. This explains the growing interest in advanced techniques of malware detection [3] and malware classification [4].

Antivirus products have focused on using individual malware signatures for malware detection. More recently, with the development of obfuscation techniques [1, 2], some variants/members of a malware family are mutated using polymorphic or metamorphic techniques. As the resultant obfuscated behavior not just increases the difficulty of identifying effective signatures, malware may bypass static signature-based detection and evade virus scanners. In addition, many works, such as Sebastián *et al*. [5] and Zhang *et al*. [6], pointed out that labels (malware classes) from different antivirus vendors are inconsistent due to the classification rules of vendors. The inherent label inconsistency problem of antivirus vendors leads to profound difficulties in the study of the characteristics of malware family. All issues above suggest that a behavior-based approach is needed in investigating the characteristics of malware.

One of the major characteristics of malware is its family. The criteria of a malware family are based on common features such as attribution to the same origins or the same techniques that can be observed from the execution activities. Reversely, a malware has common behaviors inherited from its malware family, while it may exhibit distinct behavior [6]. Thus, learning from the behavior of each individual variant of a family helps to understand the scope of a malware family, and to detect or classify unseen variants of the malware family. Malware detection, another popular cybersecurity application, determines whether the examined software is a malware or not, which also requires knowing the common characteristics of malware to make the decision.

Representation learning learns the characteristics of input data (such as execution traces) and transforms to a low-dimensional representation [7] using neural networks with little or no domain knowledge that the learned representation can be for later uses, such as malware detection, classification or clustering. Representation learning has been shown a great success in the field of Artificial Intelligence, especially in Natural Language Processing (NLP) [8–10]. The representation learning of cybersecurity emerged, such as binary code similarity detection [11], cyber-attack [12], and static malware analysis [6, 13]. However, the representation learning of common behaviors from dynamic malware execution traces has rarely been investigated.

Inspired by the recent advances in the deep representation learning, this paper introduces a malware representation learning framework, composed of two components, RASMMA and RASNN, accounting for the common characteristics of a set of labeled malware samples. First, a malware sample is executed in a VM or sandbox [14, 15] and its dynamic execution trace, represented as a sequence of API calls with its parameters and return values is collected. Second, the collected execution trace is used to extract the common behaviors of a family by using the RASMMA algorithm. Finally, the output from RASMMA is fed into the proposed neural network model, RASNN, to capture the invariant representation of the set of malware. The learned representation can be used for cybersecurity tasks, such as malware family classification. For the case of family classification task, when an unknown malware sample is given, its representation can be calculated from the learned representation and compared with the representations of malware families. Sufficient matching score can identify the malware family that the sample belongs. The advantage of invariant representation is that it needs to be trained one time for various application with fine-tuned only.

This paper makes the following contributions.

- We employ the RASMMA algorithm, a motif-based mining algorithm, to extract common behavior from a collection of run-time API call sequences of malware.
- We develop a neural network model, RASNN, to learn a malware representation preserving the invariant characteristics of a malware family from the output of RASMMA.
- We design an embedding function to preserve the semantics of API parameters as well as temporal relationships of relevant events in a malware run-time trace.
- We conduct an empirical study on a cybersecurity task, malware classification, to examine the efficacy of the proposed framework. It performs well that indicates that the learned representation is effective and explainable with physical meaning.

2. PROPOSED FRAMEWORK

Fig. 1 depicts the workflow of the proposed framework with components for realizing each step of our proposed approach. The proposed approach consists of the following steps.

- *Malware Profiling*. To obtain the execution profiles of malware, we employ an automatic dynamic malware profiling system [14, 15] for preventing malware evasion. As a malware is composed of several processes collaboratively conducting malicious intents, we record trace for each process to observe its individual behavior. This yields the input data for *Common Behavior Mining* in Section 2.2 and *Neural Representation Learning* in Section 3.
- *Common Behavior Mining*. The run-time API call sequence-based motif mining algorithm (RASMMA), extracts the common behaviors of a collection of labeled malware samples. RASMMA, introduced in Section 2.2, is a clustering algorithm which aligns and groups profiles and captures the common characteristics of the families.
- *Neural Representation Learning*. Using the recorded traces and their common behaviors, we train a neural network model (RASNN) to learn a malware profile representation as a pre-training model for the fine-tuning to construct task-specific model. Details are in Section 3.
- *Task-Specific Modeling*. When the profile of a given unknown sample is taken, the pre-trained RASNN model transforms its profile into numerical vectors as its behavioral patterns. Potential tasks, such as malware detection and malware classification, as the downstream tasks, finetunes the parameters to learn the task-specific representation. Details are in Section 4.2 respectively.

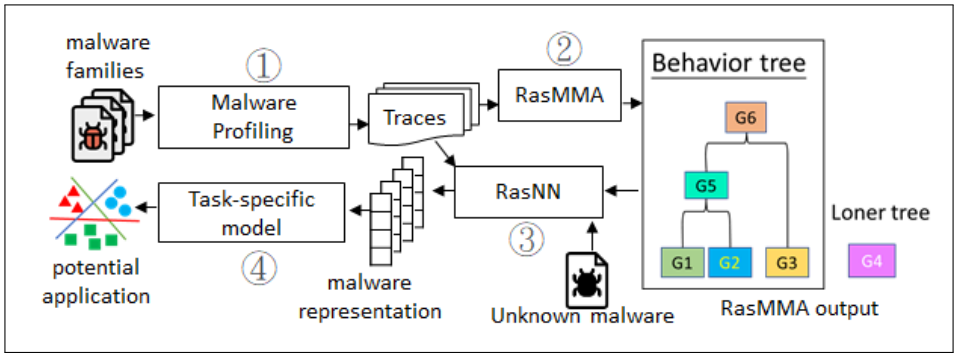


Fig. 1. Workflow of the proposed approach; ① The profiling system records malware execution traces; ② The RASMMA recognizes the common behavior patterns s of traces; ③ The RASNN learns malware representation; ④ Using malware representation, a task-specific model can be fine-tuned and produce the resulting labels for an unknown malware.

2.1 Execution Trace Generation

To capture the essentials of the execution behavior of a malware program during the course of the execution of the malware, we use an automated dynamic malware behavior profiling and analysis system based on virtual machine introspection [14, 15]. The advantage of the system is that it records only selected API calls (while it allows a user to add more API hooks) rather than all system calls, which reduces trace sizes dramatically.

Each malware is profiled for five minutes to generate the trace of Windows API calls. An execution trace contains a sequence of API call invocations of the hooked API with the parameter names and their values, as well as the returned value. Fig. 2(a) show an execution trace of a malware with MD5 value, 4ee776bc7e11de7afb1f847a8f73d445 and in Fig. 2(b), each API call in the execution trace is winnowed into a reduced format that will be fed into the following processes. For example, for the first API call LoadLibrary with the timestamp #36437000, the parameter value C:\WINDOWS\system32\icmp.dll is reduced to SYS@icmp@DLL.

2.2 Common Behavior Mining

A hierarchically clustering algorithm, named RASMMA, is used to extract common behaviors (called motif) in the form of API calls from the winnowed traces of malware variants for malware clustering. The resulting hierarchically structured clusters reveal the relationship among different malware variants from their assigned clusters. A trivial example, as shown in Fig. 1, with given common behavior and similarity score of each pair of variants, RASMMA firstly combines two variants G1 and G2 into a cluster G5 since they have the highest similarity score *th* in the first iteration, and so on RASMMA combines another variant G3 with G5 as they have the highest similarity score among all remaining pairs in the next iteration. Finally, RASMMA terminates when there is no pair of clusters with similarity score higher than a predefined threshold. The root of the resulting tree has the most common sequence of all malware variants. Note that some variants may not have common behavior with others, such as G4, called loner hereafter.

```
#364370000
LoadLibrary
lpFileName=C:\WINDOWS\system32\icmp.dll
EAX=74290000
Return=SUCCESS
#364860000
CreateFile
lpFileName =\\.\Ip
dwDesiredAccess=GENERIC_EXECUTE
dwCreationDisposition=OPEN_EXISTING
dwShareMode =FILE_SHARE_READ FILE_SHARE_WRITE
EAX=54
Return=SUCCESS
...
#367160000
RegCreateKey
hKey=HKEY_CLASSES_ROOT
lpSubKey=CLSID\{4AEDBC33-8B19-7F8D-B932-E844B2219184}\LocalServer32
EAX=0
Return=0
#367230000
RegSetValue
hKey=
HKEY_CLASSES_ROOT\CLSID\{4AEDBC33-8B19-7F8D-B932-E844B2219184}\LocalServer32
dwType=REG_SZ
lpData=malware.exe
EAX=0
Return=0
```

(a) An execution trace.

```
LoadLibrary#PR@SYS@icmp@DLL#Ret#0
CreateFile#PR@\\.\ip@NON#PR@GENERIC_EXECUTE#PR@OPEN_EXISTING#PR@FILE_SHARE_R
...
RegCreateKey#PR@HKCR@hkey_classes_root#PR@clsid\{REG}\localserver32#Ret#0
RegSetValue#PR@HKCR@clsid\{REG}\localserver32#PR@REG_SZ#PR@malware.exe#Ret#0
```

(b) The winnowed trace.

Fig. 2. An example of execution trace (a) and its winnowed trace (b) from a malware sample with MD5 value, 4ee776bc7e11de7afb1f847a8f73d445.

The idea behind RASMMA is the Global Sequence Alignment Algorithm (also known as Needleman-Wunsch algorithm) [16] for identifying the common behavior of two sequences of API calls. Given two sequences $A = (a_1, a_2, ..., a_m)$, and $B = (b_1, b_2, .., b_n)$, the matches score (α), mismatches score (β) and gaps score (γ), the score function is defined as $S(a_i, b_j) = \alpha$ if $a_i = b_j$, $S(a_i, b_j) = \beta$ if $a_i \neq b_j$ and $S(a_i, -) = \gamma$ where "$-$" is an inserted space. The Global Sequence Alignment Algorithm using dynamic programming generates a sequence $C = (c_1, c_2, .., c_l)$ with the maximum score in terms of the given score function. For instance, given two sequences ABACABDCB and DBACDCDD, and 8, –5, –3 as matches, mismatches, and gaps scores, respectively, the computation result is shown in Fig. 3 and the output sequence is BAC*DC with score 21. In general, the time complexity of the algorithm is $O(nm)$.

RASMMA includes three advantages; (a) a heuristic procedure that can combine multiple items in one iteration to boost the performance; (b) a new distance function to evaluate the distance between two text-based sequences, API calls, to find two closest variants for clustering; and (c) an alignment-based feature extraction and selection method for the representation of a cluster, to extract common behavior from a collection of malware. More details can be found in [17, 18].

A	B	A	C	A	B	D	C	B	-	
D	B	A	C	-	-	D	C	D	D	
-5	+8	+8	+8	-3	-3	+8	+8	-5	-3	=21

Fig. 3. An example for global sequence alignment.

3. NEURAL REPRESENTATION LEARNING

The purpose of the neural model, RASNN, is to compute a malware representation of representative motif sequences from a given execution profile. Two proposed modules include a) an embedder to transform each Windows API call into a numerical vector (embedding), which preserves the relations within API calls and the semantics of parameter values, and b) an encoder to process sequences of Windows API call vectors and calculate the attention weights among them to examine the importance of each Windows API call. Fig. 4 describes the neural model architecture.

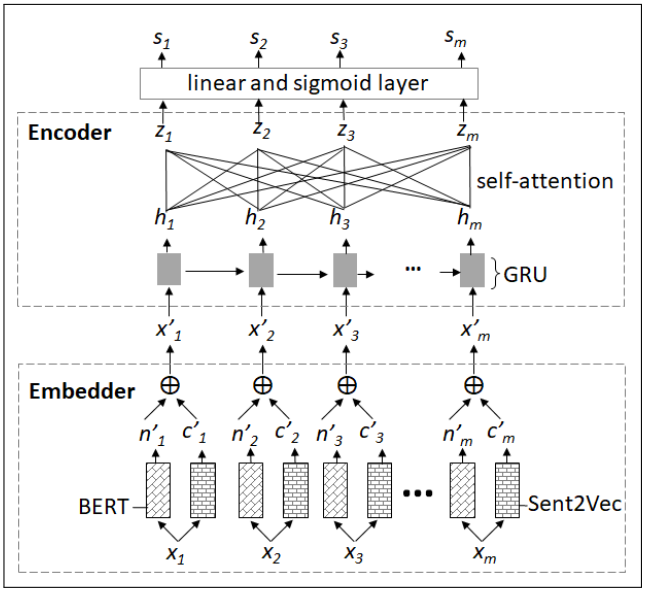


Fig. 4. Model architecture.

3.1 Task Definition

Given an input malware profile $x = \{x_1, \ldots, x_m\}$, we seek to identify its representative sequence as $s = \{s_1, \ldots, s_m\}$. More specifically, each API call in x represents each line in Fig. 2(b), its corresponding output value $s = \{s_1, \ldots, s_m\}$ denotes whether an API call x_i is representative or not, as the outcome of RASMMA. The malware representation is obtained by $y_i = \hat{s}_i \cdot x'_i$, where x' is the API call embedding from the embedder and $\hat{s}_i$ is the predicted API call's importance score from the RASNN.

3.2 Embedder

An embedder takes a variable-length execution trace $x = \{x_1, \ldots, x_m\}$ as the input and outputs a sequence of embedding vectors $x' = \{x'_1, \ldots, x'_m\}$. A Windows API call x_i consists of an API function name, one or more parameter values, and one (if any) return value. Previous work used one-hot encoding [19] or a learnable embedding matrix [20] to generate input embeddings. Since parameter and return values are considered in this work, it is difficult to adopt these approaches directly because of the infinite number of possible values. Additionally, recent advances in deep learning first pre-train a neural network on a task and then fine-tune it to yield a new purpose-specific model. Thus, the embedder uses two state-of-the-art pre-trained models – BERT [10] and SENT2VEC [9] – to build the embedding matrice.

BERT (bidirectional encoder representations from transformers) [10] is used to initialize the function name embedding n'_i. It yields promising results in many natural language processing tasks such as sentiment analysis and question answering. The key innovation of BERT is the learning of contextual relations not only between tokens in a single sentence but also between two contiguous sentences. The BERT model is used as the basis for our embedder and is fine-tuned on our dataset. This allows the embedder to enhance the semantic structure of a single Windows API call. Moreover, it yields a better understanding of the relationships between two consecutive API calls. Here, we take a single profile as a document and a single Windows API call as a sentence. After the training, we extract the last four hidden states of the function names and average them to populate the learnable embedding matrix E as

$$n'_i = En_i \tag{1}$$

where a parameter matrix $E \in R^{d \times |n|}$, where $|n|$ denotes the number of selected function names, and d denotes the BERT embedding size.

SENT2VEC [9] is an unsupervised learning model for sentence embeddings in the field of NLP. The basic concept behind which is that it considers not only unigrams but also n-gram sequences in a single sentence, allowing the model to learn the sentence embedding based on its possible constituent words.

We treat a Windows API call as a sentence and use SENT2VEC to learn API call, including parameters and return values, embeddings. First, a Windows API call x is changed to lowercase and tokenized. The SENT2VEC model learns each token and n-gram embedding within a single Windows API call. The API call embedding c'_i is then the average of the n-gram embeddings.

$$c'_i = \frac{1}{|R(x)|} \sum_{w \in R(x)} c'_w \tag{2}$$

where $R(x)$ is the list of n-grams in a given Windows API call x, $c' \in R^d$ represents the learned source embedding.

Both BERT and Sent2Vec treat an API call as a sentence, tokenize it with the delimiters, and learn the contextual relations among a function name, parameter tokens, and a return value of an API call. Both of them are trained on our dataset separately and frozen for the following procedure. Each API call embedding x'_i is composed of a function name

embedding n'_i and a complete API call embedding c'_i:

$$x'_i = n'_i \oplus c'_i \tag{3}$$

where $\oplus$ is element-wise addition.

3.3 Encoder

A malicious execution profile comprise sequences of interleaved API calls with different intentions, namely, normal behaviors, redundant events, and malicious actions. Also, among Windows API calls there exist several types of relations. For instance, a cross-reference relation (*e.g.*, two API call may use the same parameter value) suggests the associated API calls are of the same intention. This can complicate the identification of representative API call sequences.

With the encoder, we seek to process each API call embedding $x' = \{x'_1, \ldots, x'_m\}$ in an execution profile and produce a series of importance scores $s = \{s_1, \ldots, s_m\}$. The encoder accounts for this using a GRU layer and a self-attention layer. The GRU layer processes the ordinal relation over API call embeddings, the self-attention layer calculates the association between API call embeddings, and the sigmoid layer outputs the ratio between the probability that a certain API call is important and not. Gated recurrent unit [21] is part of the recurrent neural networks (RNNs) family and is used to learn a hidden state h_i at timestep i which can be seen as a summary of the past sequence from the beginning up to $i-1$ and the current input x'_i.

$$h_i = GRU(x'_i, h_{i-1}) \tag{4}$$

This summary maps an arbitrary length sequence $h_1, \ldots, h_{i-1}$ and the current observation x'_i to a vector h_i. Compared to other RNNs, GRU solves the vanishing gradient problem [22] with equally fair results.

Next, we utilize a self-attention layer to compute the weighted API call latent variable z_i over the hidden state h_i from the GRU layer. The attention weights are calculated by comparing each pair of Windows API calls in Eq. (5), and normalized with the softmax function Eq. (6) to produce a distinct distribution for each API call in a profile. Finally, these weights are then multiplied with the associated hidden state to obtain the self-attended API call representations in Eq. (7).

$$e_{ij} = sigmoid\left(W_a \, tanh\left(h_i^T W_i + h_j^T W_j + b_i\right) + b_a\right) \tag{5}$$

$$a_i = softmax(e_i) \tag{6}$$

$$z_i = \sum_j a_{ij} h_j \tag{7}$$

where W_i, W_j, and W_a are weight matrices, b_i and b_a are distinct bias. A self-attended API call representation z_i can be seen as the weighted summarization with respect to the API call input i over the API call representations. The self-attention mechanism compares the relations of each pair of API calls and captures their relative importance. This is an improvement of previous work [20] which focuses only on consecutive API call sequences in a given profile.

Finally, the representation Z is then passed to a feed-forward layer, which consists of a linear layer and a sigmoid layer. The sigmoid layer outputs the conditional probabilities of the estimation, called importance score $\hat{s} = \{\hat{s}_1 ... \hat{s}_m\}$.

$$\hat{s} = sigmoid(ZW^z) \tag{8}$$

where $W^z \in R^{|h|}$.

3.4 Training

The output from RASMMA is the ground truth for the importance scores s, $s \in \{0,1\}$. We train the model using binary cross-entropy loss.

$$L = -\frac{1}{n}\sum_{i=1}^{n} s_i \log(\hat{s}_i) + (1-s_i)\log(1-\hat{s}_i) \tag{9}$$

where n is the number of profiles in a training dataset. The Adam optimizer [23] was used, and early stopping [24] was employed on the validation dataset to prevent overfitting.

4. EVALUATION

The goal of our evaluation is to examine whether the proposed components from RASNN is effective and whether it can be applied in other cybersecurity tasks.

4.1 Component Effectiveness

For evaluating the effectiveness of the proposed RASNN, we are interested in answering the following question:
Q1: How effectively does the proposed RASNN identify important API call sequences?

4.1.1 Dataset

We selected 6,585 malware samples from NCHC[1], Taiwan. Using the VMI profiling system, we generated a profile per process, and obtained 9,819 profiles from the samples. We uploaded the samples to the VirusTotal[2] website in May, 2019 to obtain their labels. VirusTotal returned family labels defined by various anti-virus vendors. However, the family labels are inconsistent. For example, the malware with MD5 value, 0d885d060776b823c9cf039695991731, is labeled WORM/Vobfus.CF, Gen:Variant.Chinky, and Win32/AutoRun.VB.AGQ, by Avi-ra, BitDefender, and ESET-NOD32 respectively. Anti-virus vendors each have their own naming schemes, and thus often can not reach a consensus, as described in [5, 6]. We determined each family label by a majority vote of vendors.

We follow [18] to set the RasMMA similarity threshold th as 0.8 to compute RasMMA behavior trees, and further removed samples with too many or too few API calls and the final dataset included 8,176 profiles from 6,056 samples, and 133 families yielding 808 behavior subtrees. We randomly divided the dataset into a training dataset (80%),

[1] https://owl.nchc.org.tw/
[2] https://www.virustotal.com/gui/home/upload

Table 1. Descriptive statistics for training, development, and testing sets.

	Training	Validation	Test
Samples	5,466	431	481
Profiles	7,215	453	508
Families	133	47	47
Subtrees	808	135	135

a validation dataset (10%), and a testing set (10%). The numbers of samples, profile, families and subtrees are shown in Table 1.

4.1.2 Implementation details

We set the sizes of the embedding unit n', c' to 768 and the hidden unit h, k, q, v to 192. We used the Adam optimizer with a learning rate of 0.001. The mini-batch size for the update was set to 128.

4.1.3 Results

In the RASNN, the embedder employs both the function name embedding from BERT and the complete API call embedding from SENT2VEC, and the encoder uses both the GRU and self-attention. Ablation tests are used to investigate the contribution of each component in the neural model to overall performance. In the experiment, we compare with the embedder using BERT or Sent2Vec only. For the encoder, we present GRU and multi-layer perceptron (MLP) layer only. The MLP has two fully connected layers without any recurrent neural units or attention mechanism.

We report our result on the test dataset with Precision, Recall, and F1 score. Precision denotes the ratio of correctly estimated important Windows API calls. Recall denotes the ratio of correctly estimated important Windows API calls to all profiles, reflecting the classifier's ability to detect all important API calls. F1 score denotes the weighted average of Precision and Recall. Table 2 shows the performance for each model. The proposed RASNN outperforms all other models by a large margin, showing that the proposed embedder and encoder facilitate representative API call detection. The model clearly improves significantly when considering API calls with parameters and return values, as it explicitly provides more information. In addition, the performance drops when the encoder only considers either GRU or MLP, suggesting that the learned self-attention and GRUs play an important role in capturing the relative importance and preserving the ordinal information of the Windows API calls in a profile.

Table 2. Experimental results.

	Precision	Recall	F1 score
RasNN	88.15%	86.56%	87.35%
- BERT	85.81%	86.00%	85.91%
- Sent2Vec	70.95%	85.12%	77.37%
- Self-attention	77.99%	40.01%	52.89%
- GRU	63.93%	73.13%	68.22%

These findings answer our question Q1 in that considering API calls with parameter and return values, the attention (*i.e.*, relative importance) and the ordinal information of API calls all together improve the overall performance of detecting significant behavior patterns.

4.2 Malware Classification Evaluation

Section 4.1.3 demonstrated that the pre-trained malware representation outperformed the other ablated models, the pre-trained parameters can be used to initialize a downstream task. In this section, we use malware classification to demonstrate an extension task with the pre-trained malware representation. For malware classification, when given a program, the goal of family classification is to predict which known families it belongs to. This task-specific model is formed by incorporating the pre-trained malware representation with additional RNNs and a softmax layer. Therefore, only a limited number of parameters in the task need to be trained from scratch. For evaluating the extensibility of the proposed RASNN, we are interested in investigating a research question:
Q2: How effectively is the pre-trained malware representation used to perform as a malware classification?

4.2.1 Experimental setting

For malware classification, we design two datasets to investigate the RASNN's ability in identifying common and uncommon families. One is 10 families with sufficient samples, the other one is the 10 families with uncommon families (insufficient samples). For the first set, we selected 3,781 samples from 10 families, *allaple, fakealert, fakeav, kazy, solimba, sytro, ursu, virut, vobfus, zbot*, from known malicious samples. For the second set, we additionally selected 369 samples from *bdmj, directdow, expiro, fesber, hotbar, picsys, vilsel*. They were randomly assigned 80%, 10%, 10% of malware families into the training, validation, and testing set. A neural malware classifications [25] , a DL-SVM classification for processing malware binaries, is trained using labeled data in our dataset for comparison.

4.2.2 Results

As Table 3 shown, the fine-tuned RASNN outperformed the existing neural system [25] no matter the samples are sufficient or not. For example, *sytro*, RASNN can correctly identify more samples in the family than [25] does. One of the possible reasons is that RASNN can successfully recognize the common behavior of the family, which *sytro* mainly drops copies of itself to a temporary folder , and modifies a registry key to cause a system fault. Moreover, RASNN can successfully identify uncommon samples from the test set, while [25] is sensitive to some significant but untypical behaviors. For instance, a sample (MD5: 521d02c5aad5eb050a98dd63d323d400) from *fesber* shares the same behavior with *sytro*, which creates and writes an executable file in the location of user folder. But [25] fails since it lacks the common characteristics of a malware family. This suggests that the task-specific RASNN is more robust and effective, compared to the existing end-to-end malware classification.

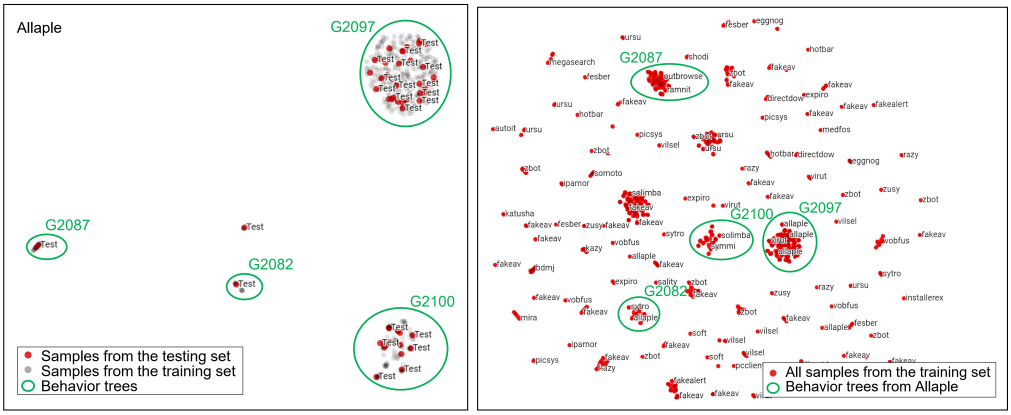
To better understand the neural malware representation, we selected a malware family, Worm:Win32/Allaple, as a case study. According to [26–28], Allaple is the name of

Table 3. Results of malware family classification.

	10 Family			10 Family + Uncommon samples		
	Precision	Recall	F1 score	Precision	Recall	F1 score
DL-SVM [25]	52.70%	44.05%	47.78%	42.28%	40.46%	40.06%
RasNN	58.69%	57.89%	57.21%	54.68%	56.02%	54.68%

a malware family that copies itself multiple times to hard drive and hijacks a COM reference for every copy of itself. It also performs denial-of-service (DoS) attacks against targeted remote Web sites. We visualize the malware representations to investigate the meanings of malware representations in the family and the comparison with malware representations from other families, and to discuss the characteristics of loner trees.

Fig. 5(a) presents the malware representations extracted from each sample in Allaple. Based on the output from RASMMA, only four behavior trees are generated, which exactly correspond to the four major groups in the figure. Most of the testing profiles are located near those from the training dataset. This demonstrates that the pretrained malware representation could capture the main characteristics of a malware family and be used to detect an unknown sample based on similar known representations. We also provide a semantic description of each group in Table 4. It is obvious that G2082 and G2097 are respectively corresponding to malicious behaviors, self-propagating and COM hijacking. G2087 and G2100 might refer to actions the malware would try to figure out the system environment before establishing any execution. This illustrates that the model seems to learn the hidden information behind behavior trees in both numerical and semantic spaces.



(a) (b)

Fig. 5. Malware representations; (a) 2D embedding of malware representations from Allaple family; The points (gray from training dataset, red from test dataset) are grouped by behavior trees; (b)2D embedding of all malware representations from training dataset.

Fig. 5(b) shows all malware representations in the training dataset, in which many profiles in the different families overlap within the major four behavior groups in the Allaple family. This meets our assumption, which is different malware families may share the same characteristics. This could be due to pluralism, that is, the characteristics

Table 4. Semantic description of four clusters.

Group	Extracted sequences
G2082	A large number of functions to create files named as a crack game executable file or a program password generator in the temporary directory "%TEMP%" in Windows, and to copy itself (malware) to the created files.
G2087	A number of functions to query registry values to collect information about RPC and functions to create endpoints for RPC over SMB.
G2097	A series of functions to access registry for registering itself (malware) as a CLSID reference to a binary path in the data field of "LocalServer32."
G2100	A series of functions to access registry for query information about "WinSock2" on the system.

of a behavior tree extend across more than one family. For example, the *Graftor*, *Virut*, and *Expiro* families share the G2097 behavior tree, which means some of them would performance COM hijacking. This evidence is consistent with the technical description of threat intelligence in [29–32]. Thus, compared to a end-to-end family classification, our classification criterion relies on the characteristics of a family. This could break the limitation when a family may exhibit more than one type of behavior, and that some of these behaviors are shared by different families.

These findings answer our question Q2 in that the malware representation can represent common behaviors of a malware family and can be used to recognize the members of a malware family. This learned behavior can effectively distinguish variants from different malware families. Thus, the proposed framework is effective for malware classification.

5. RELATED WORK

5.1 Neural Malware Analysis

A large body of literature exists in malware behavior analysis [3, 4]. This can be classified into static analysis and dynamic analysis. Static analysis collects information from binaries or source code by decompressing or unpacking rather than executing them. Dynamic analysis focuses on execution activities from API calls or system calls when a device has been infected. We focus on related work that use neural networks to model and extract malware behavior.

Malware analysis with deep neural networks, transforming various features into a numerical vector, were demonstrated that could reach high scalability and better accuracy. Such as Athiwaratkun *et al*. [33] and Pascanu *et al*. [19] investigated echo state networks (ESNs), recurrent neural networks (RNNs), convolutional neural networks and long short-term memory (LSTM) to learn mailicous events for malware classification. Agarap [25] integrated deep learning algorithms with Support Vector Machine (SVM), *i.e*. CNN-SVM, GRU-SVM and MLP-SVM to find the relation between a given malware and its corresponding family. Agrawal *et al*. [34] demonstrated the usefulness of parameters associated with Windows API calls and developed an algorithm integrated into neural networks. Huang *et al*. [20] and Yakura *et al*. [35] incorporating attention mechanism into neural networks, can respectively capture local event patterns and suspected

code segments in malware.

We differ from all of those approaches in three main ways. First, we make use of the pre-trained model to fine-tune embedding of Windows API calls. Second, we consider the ordinal and association among Windows API calls, both of which showed significantly improve recognition of the characteristics of malware. Finally, we consider to learn the importance of Windows API call sequences as the representation of a malware. It is not like most current malware analysis systems, focusing on developing end-to-end neural network models to detect whether a sample is malware, or identify which malware family it belongs to.

5.2 Representation Learning in Security

Many powerful representation-learning algorithms implemented in different data sources and domain, such as WORD2VEC [8], SENT2VEC,PAGLIARDINI2017UNSUPERVISED and BERT [10] in natural languages, and Gemini [11] and ATTACK2VEC, SHEN2019ATTACK2VEC in security, ASM2VEC,DING2019ASM2VEC and ANDRE [6] in malware analysis. We focus on related work that applies deep representation to security problems.

Embedding application has been developed in several security tasks, such as vulnerability search, intrusion prevention system, and malware detection. Xu *et al*. [11] firstly proposed a neural network-based approach, Gemini, to transform the control flow graph of each binary function into a numeric vector. This was demonstrated that it worked successfully for vulnerability search in firmware images. Shen *et al*. [12] introduced a temporal word embedding algorithm, ATTACK2VEC, to model and monitor the evolution of cyber-attacks. This can be effectively used to flag any emerging attack before unfolding. Ding *et al*. [13] developed an assembly code representation learning model, ASM2VEC, to model the control flow graph and assembly code syntax. This static approach demonstrated that learning lexical semantic information behind assembly functions could be more resilient to code obfuscation and complier optimzations. Zhang *et al*. [6] designed Android Network Representation Learning (ANDRE) model to embed heterogeneous information, including Android code sequence, metadata and label information, into a latent feature space in the network for clustering weakly-labeled malware.

The differences with related work can thus be summarized as follows: First, we mine the behavior patterns of known malware and learn them as malware representations. Second, the representation only considering dynamic features of malware can distinguish one malware's characterises from another.

6. CONCLUSION

We introduced a novel approach for generating malware representation that learning embeddings of behavior patterns as malware characteristics. Our evaluation shows that the proposed neural framework outperforms other components by large margins with respect to utilize pre-trained models to extract the properties of a single API call as well as consecutive API calls, and use GRU and self-attention to reflect dependencies among API calls. We show that pre-trained representations reduce the need for many heavily-engineered task-specific architectures. Our demonstration of a potential application, mal-

ware classification, shows that malware representation can successfully distinguish a malware from other malware families on the basis of embedded characteristics that match known malware families.

This study has the undeniable merit of offering valuable insights into malware representation, but it has some limitations. First, like supervised learning-based approaches, our framework requires malware samples for learning the characteristics of malware families. The more samples are given, the more accurate and robust representation is. Second, we define the malware representation based on the malware behavior patterns from families, the results of malware classification are acceptable but less than 60%. To improve the performance, we plan to explore significant behavior with self-supervised learning or unsupervised learning approaches, rather than the common behavior of a malware family solely. Moreover, we further consider to employ more other downstream tasks, such as behavior detection in the future. Finally, unfortunately, for most of the other malware analysis approaches were not publicly release. It was difficult to make a reasonable comparison. We further plan to release our dataset and trained model for future evaluation.

ACKNOWLEDGMENT

We thank Ying-Ren Guo for assistance with the experiments. This work was supported in part by CITI, Academia Sinica, and by MOST under grants 110-2218-E-001-001-MBK, 109-2221-E-001-010-MY3, and 109-2221-E-004-007-MY3.

REFERENCES

1. J.-M. Borello and L. Mé, "Code obfuscation techniques for metamorphic viruses," *Journal in Computer Virology*, Vol. 4, 2008, pp. 211-220.
2. I. You and K. Yim, "Malware obfuscation techniques: A brief survey," in *Proceedings of IEEE International Conference on Broadband, Wireless Computing, Communication and Applications*, 2010, pp. 297-300.
3. Y. Ye, T. Li, D. Adjeroh, and S. S. Iyengar, "A survey on malware detection using data mining techniques," *ACM Computing Surveys*, Vol. 50, 2017, pp. 1-40.
4. E. Gandotra, D. Bansal, and S. Sofat, "Malware analysis and classification: A survey," *Journal of Information Security*, Vol. 5, 2014, pp. 56-64.
5. M. Sebastián, R. Rivera, P. Kotzias, and J. Caballero, "Avclass: A tool for massive malware labeling," in *Proceedings of International Symposium on Research in Attacks, Intrusions, and Defenses*, 2016, pp. 230-253.
6. Y. Zhang, Y. Sui, S. Pan, Z. Zheng, B. Ning, I. Tsang, and W. Zhou, "Familial clustering for weakly-labeled android malware using hybrid representation learning," *IEEE Transactions on Information Forensics and Security*, Vol. 15, 2019, pp. 3401-3414.
7. Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 35, 2013, pp. 1798-1828.
8. T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv Preprint*, 2013, arXiv:1301.3781.

9. M. Pagliardini, P. Gupta, and M. Jaggi, "Unsupervised learning of sentence embeddings using compositional n-gram features," *arXiv Preprint*, 2017, arXiv:1703.02507.
10. J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv Preprint*, 2018, arXiv:1810.04805.
11. X. Xu, C. Liu, Q. Feng, H. Yin, L. Song, and D. Song, "Neural network-based graph embedding for cross-platform binary code similarity detection," in *Proceedings of ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 363-376.
12. Y. Shen and G. Stringhini, "Attack2vec: Leveraging temporal word embeddings to understand the evolution of cyberattacks," in *Proceedings of the 28th USENIX Security Symposium*, 2019, pp. 905-921.
13. S. H. Ding, B. C. Fung, and P. Charland, "Asm2vec: Boosting static representation robustness for binary clone search against code obfuscation and compiler optimization," in *Proceedings of IEEE Symposium on Security and Privacy*, 2019, pp. 472-489.
14. S.-W. Hsiao, Y.-N. Chen, Y. S. Sun, and M. C. Chen, "A cooperative botnet profiling and detection in virtualized environment," in *Proceedings of IEEE Conference on Communications and Network Security*, 2013, pp. 154-162.
15. S.-W. Hsiao, Y. S. Sun, and M. C. Chen, "Hardware-assisted mmu redirection for in-guest monitoring and api profiling," *IEEE Transactions on Information Forensics and Security*, Vol. 15, 2020, pp. 2402-2416.
16. S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of Molecular Biology*, Vol. 48, 1970, pp. 443-453.
17. W.-J. Chiu, "Automated malware family signature generation based on runtime api call sequence," Master's Thesis, Department of Information Management, National Taiwan University, 2018.
18. S. C. Chang, Y. S. Sun, W. L. Chuang, M. C. Chen, B. Sun, and T. Takahashi, "Antsdroid:using rasmma algorithm to generate malware behavior characteristics of android malware family," in *Proceedings of IEEE 23rd Pacific Rim International Symposium on Dependable Computing*, 2018, pp. 257-262.
19. R. Pascanu, J. W. Stokes, H. Sanossian, M. Marinescu, and A. Thomas, "Malware classification with recurrent networks," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, 2015, pp. 1916-1920.
20. Y.-T. Huang, Y.-Y. Chen, C.-C. Yang, Y. Sun, S.-W. Hsiao, and M. C. Chen, "Tagging malware intentions by using attention-based sequence-to-sequence neural network," in *Proceedings of Australasian Conference on Information Security and Privacy*, 2019, pp. 660-668.
21. K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv Preprint*, 2014, arXiv:1409.1259.
22. Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, Vol. 5, 1994, pp. 157-166.

23. D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv Preprint*, 2014, arXiv:1412.6980.
24. R. Caruana, S. Lawrence, and L. Giles, "Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping," *Advances in Neural Information Processing Systems*, 2001, pp. 402-408.
25. A. F. Agarap, "Towards building an intelligent anti-malware system: a deep learning approach using support vector machine (svm) for malware classification," *arXiv Preprint*, 2017, arXiv:1801.00318.
26. Microsoft, "Microsoft security intelligence: Win32/Allaple," https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?name=Win32%2FAllaple.
27. F-Secure, "F-secure threat description: Net-worm:W32/Allaple.A," https://www.f-secure.com/v-descs/allaple_a.shtml.
28. Sophos, "Sophos threat analyses: W32/Allaple-F," https://www.sophos.com/en-us/threat-center/threat-analyses/viruses-and-spyware/W32Ãllaple-F/detailed-analysis.aspx.
29. Lavasoft, "Malware descriptions: GEN.VARIANT.GRAFTOR.123377_2D16F4ED04," http://secure.lavasoft.com/mylavasoft/malware-descriptions/blog/GenVariantGraftor1233772d16f4ed04.
30. T. Micro, "Threat encyclopedia: PE_VIRUT.XQ-4," https://www.trendmicro.com/vinfo/us/threat-encyclopedia/malware/pe_virut.xq-4.
31. McAfee, "Threat Intelligence: W32/Expiro!07209D544F52," https://www.mcafee.com/enterprise/en-us/threat-intelligence.malwaretc.html?vid=3370194.
32. Adware, "Malware descriptions: Win32.Expiro.Gen.4_c78ba9cfd9," https://www.adaware.com/malware-descriptions/blog/Win32ExpiroGen4c78ba9cfd9.
33. B. Athiwaratkun and J. W. Stokes, "Malware classification with lstm and gru language models and a character-level cnn," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, 2017, pp. 2482-2486.
34. R. Agrawal, J. W. Stokes, M. Marinescu, and K. Selvaraj, "Neural sequential malware detection with parameters," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, 2018, pp. 2656-2660.
35. H. Yakura, S. Shinozaki, R. Nishimura, Y. Oyama, and J. Sakuma, "Neural malware analysis with attention mechanism," *Computers & Security*, Vol. 87, 2019, p. 101592.

Yi-Ting Huang received a Ph.D. degree in Information Management from National Taiwan University in 2015, now she is a Postdoctoral Fellow of the Research Center for Information Technology Innovation at Academia Sinica. Her current research interests include malware analysis, deep learning and natural language processing in educational applications.

Ting-Yi Chen received the BS degree in Information Management with a minor in Computer Science and Information Engineering from National Central University, in 2017 and the MS degree in Information Management from National Taiwan University in 2019. His research interests include taking advantage of machine learning and neural network techniques to solve challenge problems encountered in big data, text mining, natural language processing, and information security fields.

Shun-Wen Hsiao received the BS and Ph.D. degree from the Department of Information Management of National Taiwan University in 2004 and 2012, respectively. Since 2017, he is with the Department of Management Information Systems, National Chengchi University as an Assistant Professor. His research interests are in the area of cybersecurity, malware behavior analysis, data analysis, virtualization technology, and FinTech.

Yeali S. Sun received her BS from the Computer Science and Information Engineering Department of National Taiwan University, and MS and Ph.D. degrees in Computer Science from the University of California, Los Angeles (UCLA) in 1984 and 1988, respectively. From 1988 to 1993, she was with Bell Communications Research Inc. In August 1993, she jointed National Taiwan University and is currently a Professor of the Department of Information Management. Her research interests are in the areas of Internet security and forensics, Quality of Service (QoS), cloud computing and services, and performance modeling and evaluation.