PAPER Special Section on Data Engineering and Information Management

Efficient Dynamic Malware Analysis for Collecting HTTP Requests using Deep Learning*

Toshiki SHIBAHARA[†,††a)], ***Nonmember***, **Takeshi YAGI**[†], **Mitsuaki AKIYAMA**[†], **Daiki CHIBA**[†], ***and*** **Kunio HATO**[†], ***Members***

SUMMARY Malware-infected hosts have typically been detected using network-based Intrusion Detection Systems on the basis of characteristic patterns of HTTP requests collected with dynamic malware analysis. Since attackers continuously modify malicious HTTP requests to evade detection, novel HTTP requests sent from new malware samples need to be exhaustively collected in order to maintain a high detection rate. However, analyzing all new malware samples for a long period is infeasible in a limited amount of time. Therefore, we propose a system for efficiently collecting HTTP requests with dynamic malware analysis. Specifically, our system analyzes a malware sample for a short period and then determines whether the analysis should be continued or suspended. Our system identifies malware samples whose analyses should be continued on the basis of the network behavior in their short-period analyses. To make an accurate determination, we focus on the fact that malware communications resemble natural language from the viewpoint of data structure. We apply the recursive neural network, which has recently exhibited high classification performance in the field of natural language processing, to our proposed system. In the evaluation with 42,856 malware samples, our proposed system collected 94% of novel HTTP requests and reduced analysis time by 82% in comparison with the system that continues all analyses.

key words: *infected host detection, network behavior, sequential data, recursive neural network*

1. Introduction

Malware authors or attackers always try to evade detection methods to increase the number of malware-infected hosts on the Internet. The detection methods are broadly divided into three types: static feature-, host-, and network-based. Static feature-based methods, such as general anti-virus engines, are easily evaded by changing malware samples' code structure with packing techniques [2]. Host-based methods, such as application programming interface (API) call mining [3], are evaded by blending malicious API calls into legitimate system processes with API hooking or dynamic link library (DLL) injection [4].

This arms race regarding static feature-based and host-based methods increases the importance of network-based methods such as malicious communication detection [5], [6] and blacklist-based detection. These methods are difficult to evade because malicious network behavior is definitely observed. For example, attackers need to coordinate infected hosts to accomplish their mission by distributing configuration files or sending commands from command and control (C&C) servers. Network-based methods detect communications sent from infected hosts on the basis of characteristic patterns in HTTP or HTTPS requests collected with dynamic malware analysis. Note that we consider both HTTP and HTTPS requests but hereafter refer to both as "HTTP requests" for simplicity.

Since attackers modify a part of a program of malware samples or produce totally new ones, patterns of malicious HTTP requests change continuously over time. As a result, the detection rate of network-based methods gradually degrades. To maintain a high detection rate, novel HTTP requests, which have not been collected in past analyses, are collected by analyzing new malware samples typically for a *fixed short period* such as five minutes [7]. Ideally, all malware samples should be analyzed for a *long period* to collect more novel HTTP requests. If more characteristic patterns are identified by using collected requests, they increase the detection rate and make it even more difficult for attackers to evade network-based methods. However, more than 350 million new malware samples were detected in 2016 [8], and analyzing all new malware samples for a long period is obviously infeasible in a limited amount of time. Efficient dynamic analysis is thus required to collect more novel HTTP requests in a shorter analysis time.

The efficiency of dynamic analysis can be enhanced by prioritizing analyses of malware samples that send novel HTTP requests. Identifying such malware samples before analyzing them is difficult because malware samples are obfuscated with packing techniques [2]. Therefore, we propose a system that analyzes a malware sample for a short period and then determines whether the analysis should be continued or suspended. Our system leverages only network behavior, i.e., communications, for determination because host behavior might be concealed with API hooking or DLL injection [4].

As determination methods, there are two possible approaches referring to conventional signature-based detection leveraged by anti-virus engines and API call mining. One is based on the presence of a certain communication, and the other is based on network behavior modeling. For the first approach, the presence of novel HTTP requests is assumed to be an adequate criterion. Specifically, an analysis is con-

Manuscript received May 24, 2018.
Manuscript revised November 2, 2018.
Manuscript publicized February 1, 2019.

[†]The authors are with NTT Secure Platform Laboratories, Musashino-shi, 185–8585 Japan.
[††]The author is with Osaka University, Suita-shi, 565–0871 Japan.

*An earlier version of this paper appeared in [1].

a) E-mail: toshiki.shibahara.de@hco.ntt.co.jp

DOI: 10.1587/transinf.2018DAP0001

tinued if one or more novel HTTP requests are collected in the short-period analysis. This approach is based on the intuition that such malware samples are likely to continuously send other novel requests. However, some malware samples stop their malicious activities due to failed communications with C&C servers. After that point, they no longer communicate with any hosts. Furthermore, the same malware sample might send different HTTP requests after a certain point of time if the secondary malware samples, which are downloaded by the original one, behave differently. In this case, an accurate determination cannot be made on the basis of communications sent in the short-period analysis. From these two points, the accurate determination is difficult to make on the basis of the first approach.

In the second approach based on network behavior modeling, we determine whether the analysis should be continued or suspended on the basis of network behavior in the short-period analysis. However, it is difficult to accurately predict whether novel HTTP requests will be collected by continuing the analysis. This is because this prediction requires information about attackers' attempts, such as attacks launched by malware samples, or infrastructure, such as the configuration of C&C servers. Inspired by the method optimizing the approximate loss function [9], we tackle this problem by relaxing the condition where the analysis is continued. Specifically, our system predicts the probability that a malware sample will send HTTP requests not collected in the short-period analysis and continues the analysis if the probability is high. Such prediction can be made by modeling continuous malicious activities such as secondary malware downloads and communications with C&C servers. Our system can collect many HTTP requests, which are expected to include not only ones previously collected in the past analyses but also many novel ones.

To make an accurate prediction, we focus on the fact that malware communications resemble natural language from the viewpoint of data structure. Natural language has a recursive structure; a phrase, e.g., noun phrase, consists of several words, and a sentence consists of several phrases. Similarly, malware communications have a recursive structure. A malicious activity, e.g., secondary malware downloads and communications with C&C servers, consists of several communications, e.g., DNS queries and HTTP requests. Functions of malware samples, e.g., information leakage and attack to other hosts, consist of several malicious activities. To capture such a recursive structure in malware communications, methods for natural language processing (NLP) are expected to be effective.

Many methods have been proposed for NLP: recursive neural network (RNN) [10], long short-term memory [11], and combination of word2vec and a convolutional neural network [12]. All methods can be applied to our proposed system, but we evaluated our system with a method expected to achieve high classification performance. In natural language, words comprising a phrase are adjacent to each other, but in malware communications, communications consisting of a malicious activity are not necessarily adjacent. For example, a communication for testing Internet connection can be sent between communications with a C&C server when a malware sample sends communications for periodically testing Internet connection. This makes classification difficult because methods for NLP prioritize closer words. However, the RNN [10] is expected to be unaffected by this challenge. The RNN performs classification on the basis of inferred recursive structure: a tree-structured neural network. If we construct tree-structured neural networks considering the recursive structure of malware communications, the RNN can accurately classify malware communications. Therefore, we apply the RNN to our proposed system and empirically discuss its efficiency at collecting novel HTTP requests using 42,856 malware samples collected over six months.

The main contributions of this paper are as follows.

- We propose a system that identifies malware samples whose analyses should be continued on the basis of the network behavior in their short-period analyses. In our evaluation, we show that our system can efficiently collect novel HTTP requests in a limited amount of time by keeping the number of malware samples that are analyzed for long period to a minimum.
- To the best of our knowledge, we are the first to apply the RNN to malware communication analysis and to show that it can effectively capture the characteristics of malware communications.

2. Recursive Neural Network

The RNN is a tree-structured neural network. It is used for parsing natural language sentences [13] and sentiment analysis [10] in the field of NLP. Our proposed system uses the recursive neural tensor network (RNTN) [10], which improves on the performance of the RNN by using a tensor. The tensor enables the RNTN to calculate high-order composition of input features. The RNTN is a tree-structured network similar to the RNN, as shown in Fig. 1. When the input sequence $x_1, x_2, x_3, \ldots$ is given, these inputs are assigned to leaf nodes in sequence.

Each node has an n-dimensional feature vector and a label. The feature vectors of parent nodes are calculated us-

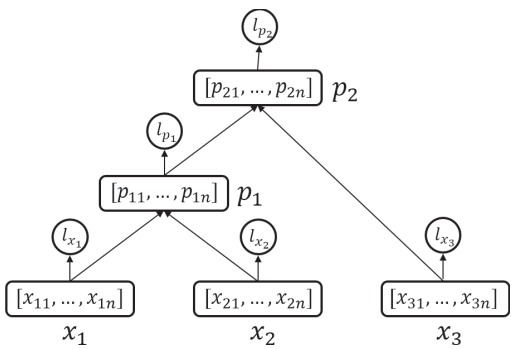

Fig. 1 Recursive neural tensor network.

ing the feature vectors of their child nodes. For example, the feature vector of p_1 is calculated using the feature vectors x_1 and x_2:

$$p_1 = f\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T V^{[1:n]} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + W \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right), \quad (1)$$

where $V^{[1:n]} \in \mathbb{R}^{2n \times 2n \times n}$ and $W \in \mathbb{R}^{n \times 2n}$. The label l_{x_1}, which is the label of node x_1, is calculated as

$$l_{x_1} = \mathrm{softmax}(W_s x_1), \quad (2)$$

where $W_s \in \mathbb{R}^{d \times n}$ and d is the number of labels. Tensor V and matrices W and W_s are commonly used in all nodes. The objective with this method is to estimate these variables. The extension of backpropagation on the basis of the prediction error of labels is applied to this process.

The feature vectors of parent nodes have two characteristics. The first is that they are calculated on the basis of the sequence of words. The second is that they represent the semantics of phrases. This enables phrases to be found that have similar meanings by finding similar feature vectors. In other words, the RNN can infer the same semantics of phrases composed of different words.

3. Proposed System

In this section, we first discuss the design of our proposed system and then describe its implementation.

3.1 System Design

To improve the efficiency of dynamic analysis, systems based on static features [14] and host behavior [15] have been proposed. However, no system based on network behavior has been proposed, but similar systems have, e.g., malware detection or classification systems based on network behavior. In system design, even if no system has been proposed for the same purpose, referring to systems proposed for a similar purpose is beneficial. The similar systems are divided into network-signature-based [5], [6], correlation-based [16], [17] and statistics-based [18], [19] systems. Network-signature-based systems detect bots by signature matching. Correlation-based systems detect botnets on the basis of the correlation of network behavior such as communications with C&C servers and attacks by infected hosts. Nevertheless, it is assumed with these systems that network behavior is collected over a long period. In particular, with network-signature-based systems, the probability of signature matching decreases as the period of network-behavior collection shortens. Analogously, with correlation-based systems, the number of communications with C&C servers and attacks decreases. Therefore, it is difficult to determine whether dynamic analysis should be continued by using these systems. On the other hand, statistics-based systems extract statistical features from not only communications with C&C servers and attacks but also other communications and classify them with machine learning. For this reason, such systems are more suitable for classification based on a short period of network-behavior collection than other systems. Therefore, we apply machine learning to our proposed system.

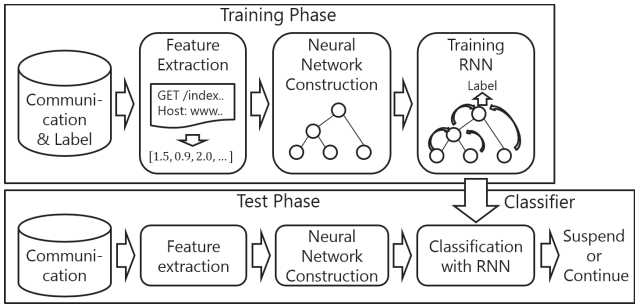

Fig. 2 Overview of proposed system.

The conventional statistics-based system [18], [19] extracts statistical features such as the number of communications for each application protocol and n-grams of network events. The effectiveness of these features also depends on the number of communications sent in the short-period analysis. If only a few communications are sent, the features will not be meaningful. To overcome this problem, we focus on the change in communication purposes. For example, a malware sample accesses a popular web page for testing the Internet connection, receives the command from the attacker, and then conducts a secondary attack. To this end, we use the sequence of communications to model malicious activities and apply the RNN to our proposed system.

Figure 2 shows the overview of our proposed system. In the training phase, our system receives pairs of malware communications and a label as input. Malware communications are collected by analyzing malware samples for a short period. The label indicates whether the analysis should be continued or suspended. In the end of the training phase, our system outputs the trained classifier of the RNN. In the test phase, our system receives new malware communications as input and outputs the determination of whether the analysis should be continued or suspended. The communications are collected by analyzing new malware samples for a short period. In both phases, a feature vector is extracted from each communication, and the neural network is constructed on the basis of communication purposes. The feature vectors and neural network are used for training and classification. We describe details of these components in the following subsections.

3.2 Feature Extraction

The first step involves extracting features from communications collected with dynamic analysis. To achieve high classification performance, we must extract features representing a part of malicious activities, e.g., Internet connection tests, updates, and command reception. To this end, we extract 7 general features, 4 hostname-based features, and 7 HTTP-based features, totaling 18 features as

Table 1 List of features.

Type	No.	Feature	Reference
General	1	Protocol	[18], [19]
	2	Elapsed time	[22]
	3	Interval	[22]
	4	Existence of the identical communication	[22]
	5	Data size of request	[19]
	6	Data size of response	[19]
	7	Port number	[19], [21]
Hostname	8	TLD rank	[21]
	9	Presence of "ip" in FQDN	[20], [21]
	10	Presence of IP address	[20], [21]
	11	Presence of subdomain	[21]
HTTP	12	HTTP method	[5]
	13	Status code	[19]
	14	Presence of `.exe` in filename	[5]
	15	Presence of `.zip` in filename	[5]
	16	Depth of file path	[5]
	17	# of query parameters	[5]
	18	User agent	[5], [22]

shown in Table 1. We design these features by referring to not only the statistics-based systems [18], [19] but also the network-signature-based system [5], malicious URL detection systems [20], [21], and malware download detection system [22].

General features are the protocol, elapsed time, interval, existence of the identical communication, data size of request, data size of response, and port number (Nos. 1–7). The elapsed time is the difference between the analysis start time and the time when the target communication is sent. The interval is the difference between the time when the last and the target communications are sent. We set 1 for the existence of the identical communication if the identical communications have already been collected in the analysis and 0 if they have not. Hostname-based features include the top level domain (TLD) rank, presence of "ip" in the fully qualified domain name (FQDN), presence of IP address, and presence of subdomain (Nos. 8–11). We use the highest Alexa[†] rank among domains with a certain TLD as the TLD rank. HTTP-based features include the HTTP method, status code, presence of `.exe` in filename, presence of `.zip` in filename, depth of file path, number of query parameters, and use agent (Nos. 12–18).

We vectorize categorical features by using one-hot encoding; we create a vector whose dimension is the number of categories and whose component corresponding to the vectorized category is 1 and other components are 0. Categorical features are protocols, HTTP methods, status codes, and user agent. Protocols are categorized into DNS, HTTP, HTTPS, other known protocols, and unknown ones. The HTTP methods are categorized into three types: GET, POST, and others. Status codes are divided into six groups on the basis of their 100 placement. For example, the first group includes 100, 101, and 102, and the second group includes 200, 201, 202, etc. The user agents are divided into three types: Mozzilla, unset, and others.

[†]https://www.alexa.com/topsites

3.3 Neural Network Construction

To capture the malicious activities, we focus on the change in communication purposes. We construct a neural network in which communications related to the same purpose compose the same subtrees. We consider the relationship between the same communication purposes and the relationship between different communication purposes. The initial situation is that each node is composed of one communication. We select a set of nodes depending on the below three criteria and create their parent nodes. The selected nodes are removed from the selection candidates, and their top node is added to the candidates. We repeat this process to construct a tree-structured neural network. The first and second criteria are the relationships between the same communication purposes. The third criterion is the relationship between different communications purposes.

Hostname. Malware samples successively send communications related to a hostname, i.e., FQDN or IP address, to accomplish a purpose such as Internet connection tests, updates, and command reception. For example, a malware sample sends a DNS query to obtain the IP address of a FQDN and then sends a HTTP request to communicate with a C&C server. We combine all communications related to the same hostname. If the number of communications is more than two, we combine communications on the basis of time ordering because they are successively sent to accomplish the same purpose. Specifically, we combine the two earliest communications and make their parent node. Then we combine the parent node and the earliest communications next to the combined ones. This process is repeated until all communications related to a certain hostname are combined.

Identical URL Path and URL Query Parameters. If communications with C&C servers fail, malware samples frequently attempt to communicate with backup or alternative C&C servers. If the communication purpose with the backup servers is the same as that with the original C&C server, the path and query of these communications are assumed to be identical. Occasionally, malware samples change the URL query value. Therefore, if the path and query parameters of two communications are identical, they are supposed to be of the same purpose. However, communications that have different purposes can have the same general path such as `/index.php`. We thus take into account communications that have one or more query parameters. If the path and query parameters of the descendant nodes of A and that of the descendant nodes of B are identical, we combine nodes A and B.

Time Difference of Communications. After nodes are combined on the basis of the first and second criteria, candidate nodes have several descendant nodes. That is to say, communications are divided into several groups of communications. If the two communication groups have depen-

dency, one group is sent subsequently to the other group. In other words, periods when communication groups are sent do not overlap. Hence, the communication groups, whose periods do not overlap, are supposed to have different purposes. Therefore, we combine two communication groups that have the closest periods. Let the set of times when communications of group A are sent be $T_A = \{t_{Ai}\}$ and let that of group B be $T_B = \{t_{Bj}\}$. The difference in periods of groups is calculated as $d = \sum_i \sum_j |t_{Ai} - t_{Bj}|$. This process is repeated until all communications become the descendant nodes of one node. In this manner, the tree-structured neural network is constructed.

3.4 Training and Classification

Since we obtain only the label of malware samples for training, we set the training labels to only the root nodes. After the training, the feature vectors of root nodes are calculated in accordance with (1). Finally, classification results are determined in accordance with (2).

4. Experimental Setup

We evaluate our proposed system in terms of efficiency for collecting novel HTTP requests. Here, we describe the experimental setup.

4.1 Dataset of Malware Samples

We conduct the evaluation with a dataset composed of malware samples collected from VirusTotal[†] in Jul.–Dec. 2017. We collect the malware samples that have different SHA1 hashes and that are detected by at least one anti-virus engine. We use AVClass [23] to refine our dataset by eliminating samples falsely detected by anti-virus engines. AVClass identifies the family name of a malware sample if several anti-virus engines label it with virtually the same family name. AVClass further identified potentially unwanted programs (PUPs) on the basis of certain keywords in labels output by anti-virus engines. We use malware samples that are identified by their family names and are not identified as PUPs by AVClass. Finally, we collected 42,856 unique malware samples.

We analyze all malware samples for 30 minutes with a safe dynamic analysis system called BotnetWatcher [24] to estimate when the determination should be made. Although we should ideally analyze them for a longer time, we selected 30 minutes as a sufficiently long time. Figure 3 shows the cumulative distribution function (CDF) of time when the last HTTP request was sent. On the basis of the data in Fig. 3, most malware samples stop sending HTTP requests within five minutes, and we can collect most HTTP requests in 30 minutes. In addition, Fig. 3 shows that it is appropriate that determination be made five minutes after analysis starts. Hence, we selected five minutes as the determination time. We label a malware sample as *suspension* if all HTTP requests, which are collected in the last 25 minutes of its analysis, are collected in the first 5 minutes. On the other hand, we label a malware sample as *continuation* if all HTTP requests, which are collected in the last 25 minutes of its analysis, are *NOT* collected in the first 5 minutes.

[†]https://www.virustotal.com/

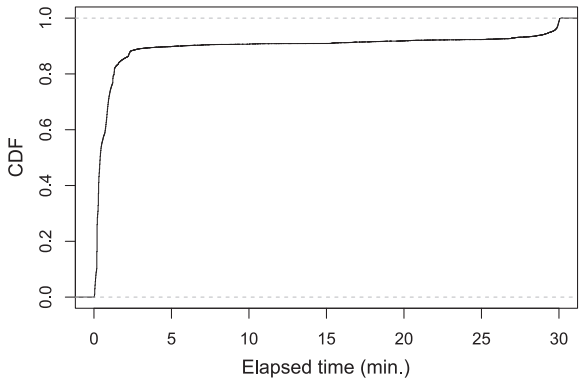

Fig. 3 CDF of time when the last HTTP request was sent.

Table 2 Dataset.

	Suspension	Continuation	Period
Training	24,988	1,003	Jul.–Oct. 2017
Test	16,335	530	Nov.–Dec. 2017

In our evaluation, we use malware samples collected during Jul.–Oct. 2017 as training data and those collected during Nov.–Dec. 2017 as test data as shown in Table 2. As mentioned above, many more samples are labeled as *suspension* than *continuation* because many malware samples stop sending HTTP requests within five minutes. Note that we analyze all malware samples for 30 minutes for labeling, but we use communications sent in the first 5 minutes of analyses for training and classification. The average number of communications in the first 5 minutes was 98.7.

4.2 Conventional Systems for Comparison

We compare our proposed system with two naive systems, the unknown request-base system, and two behavior-based systems.

One naive system, which we call the continuation system, continues all analyses. The other, which we call the suspension system, suspends all analyses. The unknown request-based system continues the analysis if one or more HTTP requests not included in the reference set of the requests are collected in the short-period analysis. We use the set of HTTP requests collected in analyses of training data as the reference set.

The behavior-based systems make a determination on the basis of network behavior in the short-period analysis. Note that our proposed system also belongs to this type of system. Our system makes a determination on the basis of a sequence of feature vectors representing communications, but other types of features vectors can be leveraged for determination. One behavior-based system, which we call the overall system, leverages feature vectors repre-

senting all communications sent in the short-period analysis. The overall system extracts feature vectors used in the statistics-based method [18]: the number of application protocols, the number of communications related to a certain domains, etc. The other behavior-based system, which we call the individual system, leverages feature vectors representing individual communications. The individual system extracts feature vectors used in our proposed system, classifies them individually, and makes a determination by integrating individual classification results of all communications sent in the short-period analysis. Specifically, if one or more communications are classified as *continuation*, the analysis is continued. If no communication is classified as *continuation*, the analysis is suspended. Random forest [25] is utilized as the machine learning algorithm for these classifications because it performs nonlinear classification with high accuracy.

4.3 Hyperparameter Optimization

We split training data into prior-training and validation data to optimize hyperparameters of the RNN. We select the combination of parameters that have the highest F-measure (see Sect. 5 for definition). For optimization, we conduct grid search by changing the batch size, initialization interval of adagrad, and learning rate. A batch size is selected from 50, 100, and 500, an initialization interval from 1, 10, and 100, and a learning rate from 0.01, 0.001, and 0.0001. We set the number of iterations for training as 100. The best combination of parameters was 100 for the batchsize, 100 for the initialization interval, and 0.001 for the learning rate.

The hyperparameters of random forest for conventional systems, i.e., the number of decision trees and the number of features for each decision tree, are optimized by using the `tuneRF` function of the `randomForest` package in R [26] when a classifier is trained. For the individual system, the number of decision trees was 473 and that of features for each decision tree was 20. For the overall system, the number of decision trees was 91 and that of features for each decision tree was 2.

5. Experimental Results

We now report the experimental results. In Sect. 5.1, we show the results of the evaluation on classification. We compare our system with two other behavior-based systems in terms of classification performance. We further investigate the effectiveness and limitations of the RNN for classification by analyzing the classifier in detail. In Sect. 5.2, we show the results of the evaluation on HTTP request collection. We compare our system with all conventional systems in terms of collection efficiency and calculation time. In Sect. 5.3, we report the network behavior of malware samples from which novel HTTP requests are successfully collected on the basis of our system but not collected on the basis of the unknown request-based system. These case studies show the effectiveness of our system for HTTP request collection.

5.1 Evaluation on Classification

We evaluate the classification performance of our system by comparing it with those of the overall and individual systems. We further analyze the classifier of the RNN in terms of the contribution of features, important behavior for classification, and false positives/negatives. In this subsection, malware samples whose analysis should be continued and suspended are referred to as *positive* and *negative* samples, respectively.

Classification Performance. We evaluate classification performance by using widely used metrics: true positive rate (TPR), false positive rate (FPR), accuracy, area under the receiver operating characteristics (ROC) curve (AUC), precision, and F-measure. Note that TPR is also known as recall. Each index is defined as follows:

$$
\begin{aligned}
&TPR = TP/(TP + FN),\\
&FPR = FP/(TN + FP),\\
&Accuracy = (TP + TN)/(TP + FN + TN + FP),\\
&Precision = TP/(TP + FP),\\
&F\text{-}measure = 2 \times TP/(2 \times TP + FP + FN),
\end{aligned}
$$

where TP denotes the number of true positives, FN false negatives, TN true negatives, and FP false positives. Note that a malware sample is classified as positive if the prediction probability is higher than a commonly used threshold, i.e., 0.5, in the training phase and evaluation of classification performance.

Table 3 shows the classification performance of each system. Our system outperformed conventional systems for most metrics. It is noteworthy that only our system achieved a high TPR and a low FPR. This resulted in our system having a high F-measure, which is defined by the number of true positives and false positives. Figure 4 shows the ROC curve from 0.0 to 0.1 FPR. Considering our dataset includes a small number of malware samples labeled *continuation*, i.e., *positive* samples, the classification performance at low FPRs is important. For example, to improve the collection rate (see Sect. 5.2 for the definition) at a TPR of 1.0, the FPR must be lower than 0.023. We selected a sufficiently wide range of FPRs for Fig. 4. Our system stably achieved a higher TPR than conventional systems in this range of FPRs. Specifically, the TPRs of the overall, individual, and proposed system at an FPR of 0.01 were 0.06, 0.454, and 0.738, respectively. The TPRs at an FPR of 0.05 were 0.366, 0.745, and 0.817. The TPRs at an FPR of 0.10 of FPR were 0.582, 0.817, and 0.847. Not only Table 3 but also Fig. 4 shows that only our system achieved a high TPR and low FPR.

Contribution of Features. To understand how the classification is conducted, the contribution of each feature to classification is informative. We evaluate how much each feature improves accuracy of the RNN. Specifically, we evaluate the accuracy of the RNN using all features and using

Table 3 Classification performance.

System	TPR (Recall)	FPR	Accuracy	AUC	Precision	F-measure
Overall	0.350	0.047	0.933	0.769	0.199	0.253
Individual	**0.794**	0.064	0.932	0.932	0.288	0.423
Proposed	0.755	**0.016**	**0.977**	**0.945**	**0.623**	**0.683**

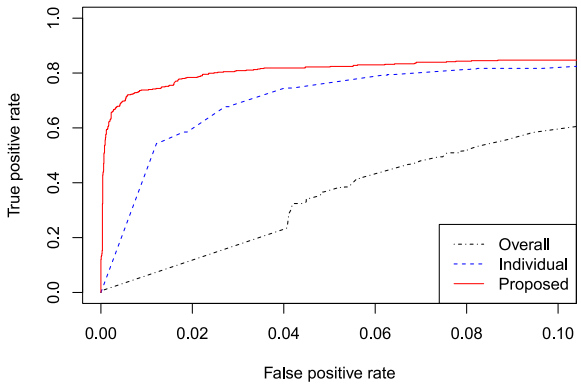

Fig. 4 ROC curve.

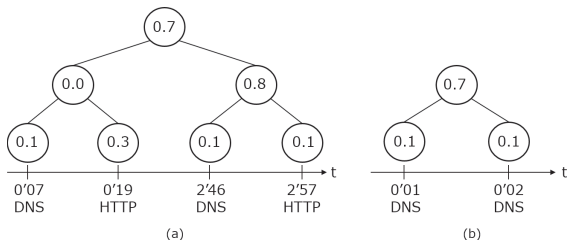

Fig. 5 Network behavior and predicted probabilities. Predicted probability is written in each node.

Table 4 Contribution of features.

Order	Type	Feature	Improvement
1	General	Elapsed time	0.068
2	General	Data size of request	0.010
3	General	Existence of the identical comm.	0.006
4	Hostname	Presence of "ip" in FQDN	0.006
5	Hostname	Presence of subdomain	0.005
6	HTTP	Depth of file path	0.004
7	General	Data size of response	0.003
8	General	Protocol	0.003
9	Hostname	TLD rank	0.003
10	HTTP	Status code	0.003

all but one of features. We then calculate their difference. Table 4 shows the ten most contributed features based on accuracy improvement. All types of features contributed to the classification, but general features contributed the most among the three feature types. These results show that application protocols other than HTTP or HTTPS are also important for classification.

Important Behavior for Classification. To more clearly identify the reason the RNN achieved the highest classification performance, we analyze important network behavior for classification. Since a small number of malware samples is labeled as *continuation* in our dataset, these samples need to be accurately detected to achieve high classification performance. For this reason, we analyze network behavior of malware samples classified as *continuation*. To this end, we take advantage of the fact that the RNN can output prediction probability of every node as well as the root node. Note that the RNN classifies malware samples as *continuation* if the prediction probability of the root node is larger than 0.5. We identify nodes satisfying the following three conditions and analyze communications that are descendants of the nodes.

1. The prediction probabilities of all descendants are lower than 0.5.
2. The prediction probabilities of all ancestors are larger than 0.5.
3. The prediction probability of the node is much larger than those of its children.

One important behavior is a pair of a DNS query and HTTP request in the latter half of the short-period analysis as shown in Fig. 5 (a). These communications are related to a FQDN with which the malware sample has not communicated before. The malware samples from which novel HTTP requests are collected by continuing their analyses successively send communications related to several FQDNs to accomplish different purposes such as Internet connection tests, updates, and command reception. On the other hand, the malware samples that stop activities in the middle of their analyses send communications related to only a few FQDNs right after starting their analyses or repeatedly send communications related to a FQDN. The RNN is supposed to learn that the behavior of starting to send communications related to a new FQDN in the latter half of the analysis indicates continuation of malicious activities.

Another important behavior is a few DNS queries right after starting the analysis as shown in Fig. 5 (b). These DNS queries are related to different FQDNs and are properly resolved. Since this malware sample sent a DNS query and HTTP request related to a different FQDN after five minutes, it is supposed to test the Internet connection with DNS and sleep a certain period of time to conceal malicious activities. On the other hand, the malware samples that stop activities in the middle of their analyses send a DNS query and HTTP request right after starting their analyses. The RNN is supposed to learn that a few successfully resolved DNS queries indicate continuation of malicious activities and are likely to be followed by other malicious activities after a certain period of time.

False Positives and False Negatives. We analyze false positives and negatives to understand limitations of the RNN. The false positives, i.e., malware samples falsely classified

Table 5 HTTP request collection efficiency.

System	Number	Ratio	Time (min.)	Reduction	Collection rate
Suspension	14,467	77%	84,325	83%	0.172
Continuation	18,676	100%	505,950	0%	0.037
Unknown request	18,550	99%	332,650	34%	0.056
Overall	14,467	77%	84,325	83%	0.172
Individual	14,467	77%	84,325	83%	0.172
Proposed	17,534	94%	94,100	82%	0.186
Oracle	18,676	100%	99,150	80%	0.188

as *continuation*, sent many DNS queries and HTTP requests in the latter half of the short-period analysis. As mentioned above, this behavior is similar to that of malware samples from which novel HTTP requests are collected by continuing their analyses. This is why false positives occur. The false negatives, i.e., malware samples falsely classified as *suspension*, sent a few HTTP requests right after starting the analyses and slept about five minutes. As mentioned above, this behavior is similar to that of malware samples that stop activities in the middle of their analyses. This is why false negatives occur. These examples show that the RNN cannot accurately classify all malware samples. However, the RNN achieved high classification performance by learning common network behavior.

5.2 Evaluation on HTTP Request Collection

We compare our system with all conventional systems in terms of collection efficiency and calculation time.

Collection Efficiency. We evaluate efficiency for collecting novel HTTP requests, which have not been collected in the analyses of training data. Table 5 shows the number of novel HTTP requests, analysis time, and collection rate, i.e., the number of novel HTTP requests per minute. The analysis time is the summation of time for which malware samples are analyzed. A malware sample is analyzed for 5 minutes if the analysis is suspended. On the other hand, a malware sample is analyzed for 30 minutes if the analysis is continued. The analysis time does not include the time of activating the dynamic analysis system [24] and for feature extraction and classification. Since the continuation system continued all analyses, its analysis time was $30 \times 17{,}302 = 505{,}950$. The analysis time of the suspension system was $5 \times 17{,}302 = 84{,}325$ because it suspended all analyses. As a reference, we also show the collection efficiency of the classifier whose accuracy is 1.0 (oracle in Table 5). We further calculate the ratio of collected HTTP requests and time reduction compared with the continuation system. In this evaluation, we optimize the threshold of prediction probability above which malware samples are classified as *continuation* because the best threshold in terms of classification performance is different from that in terms of collection efficiency. Specifically, we select the best threshold in terms of collection rate.

The unknown request-based system collected most novel HTTP requests but did not sufficiently reduce analysis time. The overall and individual system could not improve the collection rate by determination due to the low classification performance. Consequently, their best collection rate was the same as that of the suspension system. Our system could improve the collection rate by determination and thus collected a large number of novel HTTP requests in a short analysis time. Our system had a higher collection rate than the conventional systems.

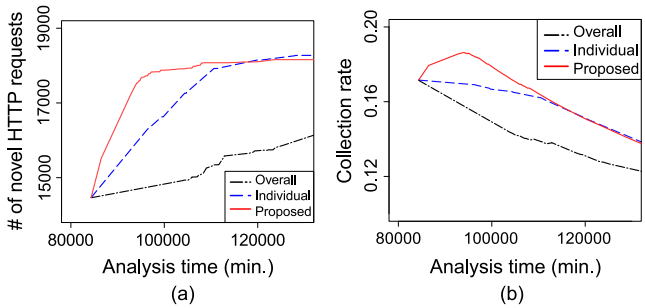

Fig. 6 HTTP request collection efficiency with different thresholds.

To precisely understand the collection efficiency of behavior-based systems, we investigate the number of novel HTTP requests, analysis time, and collection rate with different thresholds as shown in Fig. 6. When a higher threshold is used, both the number of novel HTTP requests and analysis time increase because more malware samples are classified as *continuation*. To draw Fig. 6, we analyzed all malware samples for 30 minutes beforehand. If a malware sample is classified as *suspension* we use the first 5 minutes of its analysis. On the other hand, if a malware sample is classified as *continuation* we use 30 minutes of its analysis. We calculated the number of novel HTTP requests, analysis time, and collection rate with different thresholds and plot them in Fig. 6. Since our system achieved higher classification performance, it collected more novel HTTP requests in the same analysis time than other systems and successfully increased the collection rate by determination. The overall and individual systems could not sufficiently increase the number of novel HTTP requests even with longer analysis time. This resulted in the decrease in the collection rate.

Calculation Time. Considering deployment, calculation time must be much shorter than the analysis period. Hence, we investigate calculation time for feature extraction and classification as shown in Table 6. More precisely, the calculation time is the time of executing programs for feature extraction and classification per sample in the test phase. Our proposed system took more time than the conventional

Table 6 Calculation time (sec./sample).

System	Feature extraction	Classification
Overall	0.0007	0.00001
Individual	0.0008	0.00006
Proposed	0.6730	0.00387

Table 7 Network behavior of a malware sample whose secondary malware sample behaves differently.

Time	Protocol	FQDN or URL	Novel
0'05	DNS	example1.com	
0'06	HTTP	http://example1.com/malware.exe	
0'20	DNS	example2.com	
0'30	HTTP	http://example2.com/config	
5'20	DNS	example3.com	
5'40	HTTP	http://example3.com/?sid=xxx	✓

Table 8 Network behavior of a malware sample using DGA.

Time	Protocol	FQDN or URL	Novel
0'20	DNS	random1.com	
0'30	HTTP	http://random1.com/	
3'35	DNS	random2.com	
3'40	DNS	random3.com	
6'40	DNS	random4.com	
6'45	HTTP	http://random4.com/	✓

Table 9 Network behavior of a malware sample that sleeps a long time.

Time	Protocol	FQDN or URL	Novel
0'02	DNS	example1.com	
0'03	DNS	example2.com	
25'02	DNS	example3.com	
25'03	HTTP	http://example3.com/e?xxx	✓

systems, but the calculation time was much shorter than that when a malware sample is analyzed. Specifically, a malware sample is analyzed for 5 minutes if a malware sample is classified as *suspension* and analyzed for 30 minutes if a malware sample is classified as *continuation*. Therefore, our proposed system can be deployed for efficient dynamic analysis.

5.3 Case Study

We confirm the effectiveness of our system by analyzing the network behavior of malware samples from which novel HTTP requests are successfully collected on the basis of our system but not collected on the basis of the unknown request-based system. These malware samples did not send novel HTTP requests for the first 5 minutes. This is why the unknown request-based system did not collect novel HTTP requests. The network behavior shown here is simplified due to space limitations.

Different Behavior of Secondary Malware. Our proposed system collected novel HTTP requests from a malware sample whose secondary malware sample behaves differently than in the past analysis, as shown in Table 7. This malware sent communications related to `example1.com` at first and downloaded an executable, which is supposed to be a secondary malware sample. Next, it sent communications related to `example2.com`. Since one of their URLs included `config`, the malware sample is supposed to obtain the configuration of its secondary malware sample. Communications for obtaining the configuration are typical for malware samples that continue to send communications, as with the behavior shown in Fig. 5 (a). Since the RNN properly learned such behavior, this malware sample was classified as *continuation*. Then, the secondary malware sample successively sent a DNS query and HTTP request related to another FQDN. Since the training dataset included a malware sample that performed the same malicious activities as the analyzed one, the HTTP requests to `example1.com` and `example2.com` had already been collected. However, subsequent HTTP requests were novel because their secondary malware samples or their configurations were periodically changed by attackers depending on their purpose.

DGA. The next example is a malware sample using the domain-name generation algorithm (DGA) as shown in Table 8. This malware sample sends many DNS queries related to randomly generated FQDNs using DGA and then sends HTTP requests to properly resolved FQDNs. The actual strings of `random`x in Table 8 were randomly generated by concatenating some words. Attackers use DGA to evade blocking based on blacklists of FQDNs by generating different FQDNs at different times and using a few of them. Malware samples leveraging DGA typically continue to send communications. Therefore, the RNN learned such behavior and classified this malware sample as *continuation*. Properly resolved FQDNs differ depending on the time of the analysis, and `random4.com` was used at the time of the analysis for the first time. Therefore, our system collected novel HTTP requests from this malware sample.

Long Sleep. The last example is a malware sample that sleeps a long time after sending a few DNS queries, as shown in Table 9. This malware sample sends a DNS query and HTTP request related to another FQDN about 25 minutes after sending the DNS query related to `example2.com`. This malware sample is supposed to test the Internet connection right after starting the analysis and sleep a long time to conceal malicious activities. For this reason, the unknown request-based system cannot continue the analysis. However, as shown in Fig. 5 (b), the behavior of this malware sample is common for malware samples that continue to send communications. Based on this behavior, the RNN classified this malware sample as *continuation*, and our proposed system successfully collected a novel HTTP request.

6. Discussion

We discuss our focus, the validity of the evaluation, and the limitation of our system in this section.

Our Focus. Our system would not be able to increase collection efficiency if malware samples belonging to the same

family were input. Hence, the selection of input malware samples is also important to increase efficiency. However, suitable samples can be selected by simply selecting diverse malware families. For this reason, we consider how to increase collection efficiency after malware samples are selected.

Validity of Evaluation. We propose a system for efficiently collecting novel HTTP requests to enhance detection performance of the network-based methods such as malicious communication detection [5], [6] and blacklist-based detection. In Sect. 5, we showed that our proposed system efficiently collects novel HTTP requests, but we have not investigated whether it enhances detection performance. Since the effect of our system differs depending on the methods, we cannot conduct a uniform evaluation. However, we show in case studies that our system collects novel HTTP requests useful for detection.

In terms of maintaining classification performance, the classifier of the RNN needs to be retrained periodically. Considering the computational cost of retraining, the RNN should keep classification performance high for a long period. In our evaluation, we use test data collected for two months, which is a reasonable interval for retraining, and show that our system achieved high classification performance for two months. Hence, our system is expected to maintain high classification performance by retraining at most every two months.

Limitation. If malware samples sleep a long time after dynamic analysis starts, our proposed system cannot be applied. In this case, we have two options. One is continuing analysis until malware samples send communications. The other is suspending analysis. Both decrease efficiency or exhaustiveness. However, the percentage of these malware samples is reported to be only 5.39% [27]. In our evaluation, we could not apply our proposed system only to 0.4% of malware samples. Therefore, the efficiency of our system is not significantly decreased by these malware samples.

7. Related Work

7.1 Analysis, Detection, and Countermeasure

The methods for analysis, detection, and countermeasure have been extensively studied from the following viewpoints.

Static feature: One of the main methods used by modern anti-virus software is signature-based scanning. Griffin et al. proposed a string signature generation system countering variants of malware families to reduce signature database size [28]. However, almost all recent malware samples are obfuscated by the packer, and their characteristic strings are enfolded. Therefore, malware samples should be unpacked before they are applied to signature-based methods [29]–[32].

Host behavior: On an infected host, the analysis of system call or API call events is useful to detect malware samples [33]–[35]. However, there are now malware samples that circumvent or interfere with monitoring such host-based events on the analysis environment, i.e., malware sandbox. To counter sandbox-aware malware, Kirat et al. proposed an efficient analysis system running on actual hardware [36].

Network behavior: The communications initiated by malware, e.g., C&C, are useful to build countermeasures such as blacklisting and network-based signature generation [5], [37].

7.2 Appropriate Sample Selection to Avoid Full Analysis

Bayer et al. proposed a technique that avoids analyzing the same polymorphic programs and reduces the amount of time required for analyzing malware samples [15]. To detect polymorphic malware samples, it analyzes them for a short time and finds the most behaviorally similar sample. It can successfully avoid the full analysis of about 25.3% of malware samples. Neugschwandtner et al. proposed a system leveraging both the behavioral clustering of Bayer et al. [15] and static feature-based clustering that selects the malware sample, which is most likely to yield relevant information (e.g., C&C communication), without actually running it [14]. We infer that analyzing malware samples generated by the same toolkit, e.g., polymorphic malware samples, and controlled by different attackers are also important for collecting informative communication because those malware samples may access different malicious sites. In our study, however, the time reduction of analysis was superior, although we did not eliminate polymorphic malware samples, e.g., our proposed system avoids the full analysis of 98% of malware samples and reduces analysis time by 82%.

8. Conclusion

We have proposed a system for efficiently collecting novel HTTP requests with dynamic malware analysis. Specifically, we analyze a malware sample for a short period and then determine whether its analysis should be continued or suspended. Our system identifies malware samples whose analyses should be continued on the basis of the network behavior in their short-period analyses. To make an accurate prediction, we focus on the fact that malware communications resemble those of natural language from the viewpoint of data structure. For this reason, we applied the recursive neural network to our proposed system. In the evaluation with 42,856 malware samples, our proposed system collected 94% of novel HTTP requests and reduced analysis time by 82% in comparison with the system that continues all analyses. We further found that our system effectively collected novel HTTP requests from a malware sample whose secondary malware sample behaves differently, a malware sample that uses a domain generation algorithm, and a malware sample that sleeps for a long period.

References

[1] T. Shibahara, T. Yagi, M. Akiyama, D. Chiba, and T. Yada, "Efficient dynamic malware analysis based on network behavior using deep learning," Proceedings of the 59th Annual IEEE Global Communications Conference, pp.1–7, 2016.
[2] I. You and K. Yim, "Malware obfuscation techniques: A brief survey," Proceedings of the 2010 International Conference on Broadband, Wireless Computing, Communication and Applications, pp.297–300, 2010.
[3] A. Sami, B. Yadegari, H. Rahimi, N. Peiravian, S. Hashemi, and A. Hamze, "Malware detection based on mining api calls," Proceedings of the 2010 ACM Symposium on Applied Computing, pp.1020–1025, 2010.
[4] H. Father, "Hooking windows api-technics of hooking api functions on windows," CodeBreakers J, vol.1, no.2, 2004.
[5] T. Nelms, R. Perdisci, and M. Ahamad, "Execscent: Mining for new c&c domains in live networks with adaptive control protocol templates," Proceedings of the 22nd USENIX Conference on Security, pp.589–604, 2013.
[6] D. Chiba, T. Yagi, M. Akiyama, K. Aoki, T. Hariu, and S. Goto, "Botprofiler: Profiling variability of substrings in http requests to detect malware-infected hosts," Proceedings of the 2015 IEEE Trustcom/BigDataSE/ISPA, pp.758–765, 2015.
[7] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware-analysis techniques and tools," ACM computing surveys, vol.44, no.2, 2012.
[8] Symantec, "Internet security threat report," 2017. https://www.symantec.com/content/dam/symantec/docs/reports/istr-22-2017-en.pdf.
[9] J.D. Rennie and N. Srebro, "Loss functions for preference levels: Regression with discrete ordered labels," Proceedings of the IJCAI multidisciplinary workshop on advances in preference handling, pp.180–186, 2005.
[10] R. Socher, A. Perelygin, J.Y. Wu, J. Chuang, C.D. Manning, A.Y. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," Proceedings of the Conference on Empirical Methods in Natural Language Processing, p.1642, 2013.
[11] I. Sutskever, O. Vinyals, and Q.V. Le, "Sequence to sequence learning with neural networks," Advances in Neural Information Processing Systems, pp.3104–3112, 2014.
[12] Y. Kim, "Convolutional neural networks for sentence classification," arXiv preprint arXiv:1408.5882, 2014.
[13] R. Socher, C.C. Lin, C. Manning, and A.Y. Ng, "Parsing natural scenes and natural language with recursive neural networks," Proceedings of the 28th International Conference on Machine Learning, pp.129–136, 2011.
[14] M. Neugschwandtner, P.M. Comparetti, G. Jacob, and C. Kruegel, "Forecast: skimming off the malware cream," Proceedings of the 27th Annual Computer Security Applications Conference, pp.11–20, 2011.
[15] U. Bayer, E. Kirda, and C. Kruegel, "Improving the efficiency of dynamic malware analysis," Proceedings of the 2010 ACM Symposium on Applied Computing, pp.1871–1878, 2010.
[16] G. Gu, J. Zhang, and W. Lee, "Botsniffer: Detecting botnet command and control channels in network traffic," Proceedings of the 15th Annual Network and Distributed System Security Symposium, 2008.
[17] G. Gu, R. Perdisci, J. Zhang, and W. Lee, "Botminer: Clustering analysis of network traffic for protocol-and structure-independent botnet detection," Proceedings of the 17th USENIX Conference on Security, pp.139–154, 2008.
[18] S. Nari and A.A. Ghorbani, "Automated malware classification based on network behavior," Proceedings of the 2013 International Conference on Computing, Networking and Communications, pp.642–647, 2013.
[19] A. Mohaisen, A.G. West, A. Mankin, and O. Alrawi, "Chatter: Classifying malware families using system event ordering," Proceedings of the 2014 IEEE Conference on Communications and Network Security, pp.283–291, 2014.
[20] J. Ma, L.K. Saul, S. Savage, and G.M. Voelker, "Beyond blacklists: learning to detect malicious web sites from suspicious urls," Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, pp.1245–1254, 2009.
[21] D. Canali, M. Cova, G. Vigna, and C. Kruegel, "Prophiler: a fast filter for the large-scale detection of malicious web pages," Proceedings of the 20th International Conference on World Wide Web, pp.197–206, 2011.
[22] B.J. Kwon, J. Mondal, J. Jang, L. Bilge, and T. Dumitras, "The dropper effect: Insights into malware distribution with downloader graph analytics," Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, pp.1118–1129, 2015.
[23] M. Sebastián, R. Rivera, P. Kotzias, and J. Caballero, "Avclass: A tool for massive malware labeling," International Symposium on Research in Attacks, Intrusions, and Defenses, vol.9854, pp.230–253, 2016.
[24] K. Aoki, T. Yagi, M. Iwamura, and M. Itoh, "Controlling malware http communications in dynamic analysis system using search engine," Proceedings of the 3rd International Workshop on Cyberspace Safety and Security, pp.1–6, 2011.
[25] L. Breiman, "Random forests," Machine learning, vol.45, no.1, pp.5–32, 2001.
[26] R Core Team, R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria, 2016.
[27] G.N. Barbosa and R.R. Branco, "Prevalent characteristics in modern malware," BlackHat USA, 2014.
[28] K. Griffin, S. Schneider, X. Hu, and T.-C. Chiueh, "Automatic generation of string signatures for malware detection," Proceedings of the 12th International Symposium on Recent Advances in Intrusion detection, vol.5758, pp.101–120, 2009.
[29] X. Ugarte-Pedrero, D. Balzarotti, I. Santos, and P.G. Bringas, "Sok: Deep packer inspection: A longitudinal study of the complexity of run-time packers," Proceedings of the 2015 IEEE Symposium on Security and Privacy, pp.659–673, 2015.
[30] B. Yadegari, B. Johannesmeyer, B. Whitely, and S. Debray, "A generic approach to automatic deobfuscation of executable code," Proceedings of the 2015 IEEE Symposium on Security and Privacy, pp.674–691, 2015.
[31] L. Martignoni, M. Christodorescu, and S. Jha, "Omniunpack: Fast, generic, and safe unpacking of malware," Proceedings of the 23rd Annual Computer Security Applications Conference, pp.431–441, 2007.
[32] P. Royal, M. Halpin, D. Dagon, R. Edmonds, and W. Lee, "Polyunpack: Automating the hidden-code extraction of unpack-executing malware," Proceedings of the 22nd Annual Computer Security Applications Conference, pp.289–300, 2006.
[33] A.H. Sung, J. Xu, P. Chavez, and S. Mukkamala, "Static analyzer of vicious executables (save)," Proceedings of the 20th Annual Computer Security Applications Conference, pp.326–334, 2004.
[34] L. Martignoni, E. Stinson, M. Fredrikson, S. Jha, and J.C. Mitchell, "A layered architecture for detecting malicious behaviors," Proceedings of the 11th International Symposium Recent Advances in Intrusion Detection, pp.78–97, 2008.
[35] C. Kolbitsch, P.M. Comparetti, C. Kruegel, E. Kirda, X. Zhou, and X. Wang, "Effective and efficient malware detection at the end host," Proceedings of the 18th USENIX Conference on Security, pp.351–366, 2009.
[36] D. Kirat, G. Vigna, and C. Kruegel, "Barecloud: bare-metal analysis-based evasive malware detection," Proceedings of the 23rd USENIX Conference on Security, pp.287–301, 2014.
[37] R. Perdisci, W. Lee, and N. Feamster, "Behavioral clustering of http-based malware and signature generation using malicious network

traces," Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation, pp.391–404, 2010.

Toshiki Shibahara is currently a researcher at NTT Secure Platform Laboratories, Tokyo, Japan. He is also a Ph.D. student at Osaka University. He received his B.E. degree in engineering and M.E degree in information science and technology from The University of Tokyo, Japan in 2012 and 2014. Since joining Nippon Telegraph and Telephone Corporation (NTT) in 2014, he has been engaged in research on cyber security and machine learning.

Takeshi Yagi received his B.E. degree in electrical and electronic engineering and his M.E. degree in science and technology from Chiba University, Japan in 2000 and 2002. He also received his Ph.D. degree in information science and technology from Osaka University, Osaka, Japan in 2013. He joined the Nippon Telegraph and Telephone Corporation (NTT) in 2002, where he is currently researching network security, web security, honeypots, security-data analysis based on machine learning, and security intelligence technologies such as URL/domain/IP blacklisting and reputation. He is now a senior research engineer in the Cyber Security Project of NTT Secure Platform Laboratories. He is a member of the Institute of Electrical and Electronics Engineers (IEEE) and the Institute of Electrical Engineers of Japan (IEEJ) and IEICE.

Mitsuaki Akiyama received his M.E. and Ph.D. degrees in information science from Nara Institute of Science and Technology, Japan in 2007 and 2013. Since joining the Nippon Telegraph and Telephone Corporation (NTT) in 2007, he has been engaged in research and development on network security, especially honeypot and malware analysis. He is now with the Cyber Security Project of NTT Secure Platform Laboratories.

Daiki Chiba is currently a researcher at NTT Secure Platform Laboratories, Tokyo, Japan. He received his B.E., M.E., and Ph.D. degrees in computer science from Waseda University in 2011, 2013, and 2017. Since joining Nippon Telegraph and Telephone Corporation (NTT) in 2013, he has been engaged in research on cyber security through data analysis. He won the Research Award from the IEICE Technical Committee on Information and Communication System Security in 2016 and the Best Paper Award from the IEICE Communications Society in 2017. He is a member of IEEE and IEICE.

Kunio Hato received his B.E. and M.E. degrees in information processing from Tokyo Institute of Technology in 1997 and 1999, respectively. He joined the Nippon Telegraph and Telephone Corporation (NTT) in 1999, where he was previously engaged in research and development of IP VPNs, wide area Ethernet. He is now a Senior Research Engineer, Supervisor, in Cyber Security Project of NTT Secure Platform Laboratories. He was with the Network Services of NTT communications from 2014 to 2017. He is a member of IEICE.