Electronic Preprint for Journal of Information Processing Vol.27

Regular Paper

Using Seq2Seq Model to Detect Infection Focusing on Behavioral Features of Processes

Shun Tobiyama[1,a)] Yukiko Yamaguchi[2,b)] Hirokazu Hasegawa[2,c)]
Hajime Shimada[2,d)] Mitsuaki Akiyama[1,e)] Takeshi Yagi[3,f)]

Received: November 26, 2018, Accepted: June 11, 2019

Abstract: Sophisticated cyber-attacks intended to earn money or steal confidential information, such as targeted attacks, have become a serious problem. Such attacks often use specially crafted malware, which utilizes the art of hiding such as by process injection. Thus, preventing intrusion using conventional countermeasures is difficult, so a countermeasure needs to be developed that prevents attackers from reaching their ultimate goal. Therefore, we propose a method for estimating process maliciousness by focusing on process behavior. In our proposal, we first use one Seq2Seq model to extract a feature vector sequence from a process behavior log. Then, we use another Seq2Seq model to estimate the process maliciousness score by classifying the obtained feature vectors. By applying Seq2Seq models stepwise, our proposal can compress behavioral logs and extract abstracted behavioral features. We present an experimental evaluation using logs when actual malware is executed. The obtained results show that malicious processes are classified with a highest Areas Under the Curve (AUC) of 0.979 and 80% TPR even when the FPR is 1%. Furthermore, the results of an experiment using the logs when simulated attacks are executed show our proposal can detect unknown malicious processes that do not appear in training data.

Keywords: malware, machine-learning, Seq2Seq model

1. Introduction

E-crimes such as cyber-attacks have become a serious problem for Internet users. Many recent attacks are aimed at acquiring money, thus attackers are using more sophisticated techniques. One of the most common examples is targeted attacks. In these attacks, organized attackers first draw up a strategy optimized for a target to achieve the attack's purpose, and then assail the target in various ways in accordance with this strategy. The overall attack is difficult to figure out from individual indicators, so some models divide the attack into stages [1], [2]. According to the model of the Information-technology Promotion Agency (IPA) of Japan, targeted attacks are divided into five stages [1]. In the first stage, the attackers draw up a careful plan. At this stage, they collect information of the target by using social-engineering or reconnaissance malware [3]. Then in the second stage, they infiltrate the target using targeted e-mail attacks, watering hole attacks, and so on. At this stage, multiple kinds of malware are used, such as installing more sophisticated malware along with decoy malware, like Droppers or Trojans, on infected machines by executing document files containing malicious macros or scripts. Once attackers have intruded, they attempt to install a backdoor to establish a communication path with the C&C server and then build an attack infrastructure by installing additional malware or tools via the communication path. Both the backdoor and malware are difficult to detect, because they are crafted to avoid anti-virus detections, store binary code into the memory or registry so as not to exist as a file, or inject malicious code into the memory area of a benign process to conceal their existence. Malware has also appeared that has only minimal functionality at first but is able to expand itself by downloading modules from the C&C server and deploying them in the memory [4]. Then in the fourth stage, the attackers probe the system by gathering information in the machine, stealing authentication information, or expanding the infection by using not only malware but also OS standard commands or utility tools such as `net`, `psexec` [3], [5]. Finally, the attackers take full control of the target and achieve the final goal of the attack, such as stealing credential information.

To defend the machines from attackers, many malware detection methods have been proposed. However, many of these methods use the result of static/dynamic analysis of the target file as a feature, so they are not applicable in the situation where specifying a target file is hard because attackers are already in the target machine. Thus, it is not sufficient to prevent the sophisticated attacks we described above but countermeasure after intrusion is required. However, there are few researches focusing on a countermeasure after intrusion. The purpose of our research is to detect attacks even when attackers have already intruded in a target.

1 NTT Secure Platform Laboratories, Musashino, Tokyo 180–8585, Japan
2 Nagoya University, Nagoya, Aichi 464–8601, Japan
3 NTT Security Japan, Chiyoda, Tokyo 101–0021, Japan
a) syun.tobiyama.nt@hco.ntt.co.jp
b) yamaguchi@itc.nagoya-u.ac.jp
c) hasegawa@icts.nagoya-u.ac.jp
d) shimada@itc.nagoya-u.ac.jp
e) akiyama@ieee.org
f) takeshi.yagi@ntt.com

In this paper, we propose the attack detection method applicable for after intrusion by monitoring the behavior of all processes running on the machine and estimate the maliciousness of each process. Focusing on a process running on a machine is effective for considering a countermeasure after intrusion, because an action of attackers or malware finally emerges as process behavior. We believe that malicious/benign behavior is different (e.g., read/write attempt to a specific registry, deletion of a large number of files), so it is possible to detect malicious processes by modeling and classifying the behavior of a malicious/benign process. Moreover, our proposal monitors the behavior of all processes running on a target machine and then estimates the maliciousness of processes, so it can apply to the situation when specifying a target file is hard.

In our proposal, we are modeling process behavior with Seq2Seq model, which is a DNN structure used for time-series data, to effectively recognize the feature that is contained in process behavior. We show that our proposal can detect malicious processes derived from malware by using a process behavior log that is recorded during a simulated attack performed in an experimental environment with servers and clients managed by a domain controller.

2. Related Work

There are many malware detection/classification methods that use machine-learning techniques such as Support Vector Machine (SVM), Decision-Tree, or clustering algorithms [6], [7], [8]. Tian et al. proposed a method for malware detection and malware-family classification, which uses the frequency of API call or its attribute collected by executing malware binaries on a virtual environment as a feature [8]. In recent year, methods applying DNNs to malware detection also have been proposed [9], [10], [11], [12], [13]. Pascanu et al. proposed a method to classify malware by extracting a linguistic feature from an API call sequence by using an RNN [11]. This method regards an API call sequence as a sentence and extracts an effective feature by constructing the language model of API call sequences. Wang and Yiu also proposed a malware classification method that uses a multitask Seq2Seq model [12]. In this method, an API call sequence is compressed to a fixed-length feature vector by the Seq2Seq model, which is trained to output a sequence the same as the input sequence and then output the classified label and the file-access summary of the sequence. Hardy et al. proposed the malware detection method using Stacked Auto Encoder, which uses a list of API calls that the binary imports as a feature [13].

However, in these existing methods, we have to specify a program or process to collect data before classification when we have a machine which may be infected, because a feature is extracted from a result of dynamic analysis [11], [12], or static analysis [13]. This also means that these methods are targeted for classifying a specific file.

On the other hand, few methods have been proposed which use information which does not rely on a specific file, such as information extracted from process running on the machine. Nakazato et al. proposed a method for detecting suspicious processes from the process frequency, the number of users executing the process, and the network conditions [14]. This method tries to detect the attack using an anomaly detection approach. By focusing on the features of targeted attacks such that unusual processes appear and these processes communicate with C&C servers, they judge whether a process is suspicious or not by calculating the suspiciousness degree. Some Anti-virus products have the behavior based detection module [15], [16], [17]. These modules detect suspicious behavior of a machine such as unusual file/registry access attempts using signature or machine learning based techniques. However, as new malware appears every day, and attackers continue to develop methods to avoid them, these heuristic techniques require malware analysis continuously in order to create signatures or select features for machine learning. Therefore, these technologies are still insufficient to completely prevent malware or attacks, and more research for detecting such malware with less analysis is needed.

We previously proposed a method to estimate process maliciousness that uses two DNNs (a Recurrent Neural Network (RNN) and a Convolutional Neural Network (CNN)) in multiple stages [18]. There are problems that information of the feature sequence tends to be lost as the sequence becomes longer, and the amount of evaluation data is small. Therefore, in this paper, we propose a method that doesn't rely on information extracted from file by extracting the feature from the behavior of process running on the machine with a deep learning based technique.

3. Proposed Method

3.1 Overview

In our proposal, process maliciousness is estimated by recognizing and classifying linguistic features included in process behavior. Malicious and benign processes may behave differently because of the functions they have. Malicious processes often have functions that rarely appear in benign processes, such as hiding themselves or collecting input histories. Thus, malicious and benign processes are expected to have different behaviors, such as reading or writing frequency of files and registries, access attempts, and process manipulation. Therefore, we estimate process maliciousness by extracting appropriate features from process behavior.

In our proposal, we regard event sequence as process behavior. An event sequence is constructed with some events, and an event represents an operation of the process. When considering a certain event sequence W = [`CreateFile`, `SUCCESS`, `WriteFile`, `SUCCESS`], for example, `CreateFile` and `WriteFile` represent the operation of opening a file and writing to a file, respectively, and the event sequence W represents process behavior in which a file is created and written. In this way, each event in an event sequence represents an operation of a process, and a combination of events can represent continuous operation, or behavior. Process behavior can be represented by a sequence of events executed in accordance with a purpose in the process just as meaningful sentences can be constructed by arranging words in accordance with the grammar in a language. Thus, an event sequence can be seen as a sentence, and we can extract appropriate features by applying the feature extraction method in the language processing to them.

The flow of the proposed method is shown in **Fig. 1**. We first

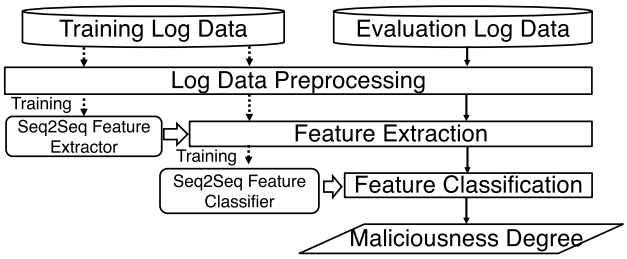


Fig. 1 Overview of our proposal.

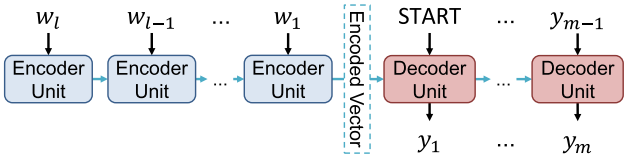


Fig. 2 Architecture of basic Seq2Seq model.

convert a process log into event sequences, which is a format that can be inputted to a feature extractor. At the training stage, we train a feature extractor and a feature classifier using converted training log data. At this time, the feature extractor is trained to output a sequence the same as an input data sequence, and the feature classifier is trained to output maliciousness of the feature vector sequence by predicting whether a sequence is benign or not. Finally, at the estimation stage, the maliciousness of processes is estimated by classifying log data of processes using the trained feature extractor and the trained feature classifier. In our proposal, we use the Seq2Seq model, which is commonly used in language processing such as machine translation, for the feature extractor and the classifier. The Seq2Seq model of the feature extractor trains linguistic features of an event sequence, and the Seq2Seq model of the feature classifier trains the maliciousness of extracted features.

3.2 Seq2Seq Model

The Seq2Seq model, which was proposed by Sutskever et al. [19], is composed of an encoder RNN to encode an input sequence to a vector and a decoder RNN to decode the encoded vector. Its basic architecture is shown in **Fig. 2**. First, the input sequence $[w_1, w_2, \ldots, w_l]$ of length l is entered to the encoder RNN in reverse order to obtain a vector in which information of the input sequence is compressed. After obtaining the vector, output is started by inputting the start symbol (START) to the decoder RNN with the compressed vector as an initial state of the RNN, and finally a prediction sequence $[y_1, y_2, \ldots, y_m]$ of length m is obtained. The input to the decoder is a correct output sequence in the training phase, and the previous output in the estimation phase. The output length m is determined manually or by the model itself. If m is determined by the model, the model should be trained to output the EOS symbol at the end of a sequence.

3.3 Preprocessing Log Data

We treat the process log recorded by Process Monitor [20] as process behavior, and use it as input data for our proposal. Process Monitor is a tool that can display and record process behavior, such as access to the file system and registry, process or thread creation, and communication with the network, in real time. Recorded information for each event is shown in **Table 1**. We create label data associated with process logs by matching a malicious process list that is first obtained by the manual analysis of processes if the logs are used for training. In our proposal, we use only values of Event, Path, and Result fields for each event. In the following description, a value of Event, Path, or Result fields are simply represented as an event, and events arranged in chronological order are represented as an event sequence. Event sequences must be preprocessed before being input to the feature extractor. We first abstract a path, then convert event sequences into vector sequences, and finally split them.

Table 1 Items recorded by Process Monitor.

Field	Description
Time	Time when the event was executed
Process Name	Process name in which event was executed
PID	Process ID in which event was executed
Event	Name of the event (ex. ReadFile)
Path	Current path when the event was executed
Result	Result of the event (ex. SUCCESS)
Detail	Further information about the event (ex. arguments of the event)

3.3.1 Path Abstraction

Process Monitor also records accessed paths when recording an event of file/registry access, so a recorded path is composed of file/directory names in file access events, or, key/subkey/value names in registry access events. To simplify the explanation, we use dirname to refer to a directory/key/subkey name, and filename to refer to a file/value name, in this section. Since the kinds of dirname and filename appearing in recorded logs become enormous and most are unique, many unique paths that do not appear in training data can appear in test data, and the number of pathkinds is unpredictable. To reduce the kinds and improve generalization performance, we abstract paths in three steps.

In the first step, we replace deeper dirnames of a path whose directory depth is deeper than three with `<omit>`. Then, we replace UUIDs appearing in filenames or dirnames with `<UUID>`. In the last step, we abstract filenames by replacing them with `<extension of filename>`.

3.3.2 Vectorization

We convert event sequences into vectors by converting each event into 1-hot vector in two steps. First, a dictionary that associates events with unique IDs is created from training data. We add `UnknownCall` to the dictionary to represent unknown events that do not appear in training data. Then each event is converted into 1-hot vector using the dictionary. A 1-hot vector of the event is filled with zeros, except for ID of the event, and its dimensions are the size of the dictionary.

3.3.3 Vector Split

In our proposal, a vector sequence is split to avoid the vanishing information problem. The Seq2Seq model has a problem that the information on the first part of the input sequence vanishes when the sequence is long. We deal with this problem by splitting the sequence and discarding the split sequences near the end. For example, when we obtain s partial sequences $[\mathbf{X'}_1, \mathbf{X'}_2, \ldots, \mathbf{X'}_s]$ by dividing the vector sequence $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N]$ of length N into subsequences whose max length is L, we only use the first S ($S \leq s$) partial sequences. Information near the top of

a sequence can be retained even when the sequence is long by extracting features from each partial sequence.

3.4 Feature Extractor Training

We use the Seq2Seq model for the feature extractor. As in the method by Wang and Yiu [12], the feature extractor is trained to output the same sequence as an input sequence. Also, in our method, the feature extractor is trained to output a vector sequence the same as the input sequence. By training in this manner, information included in a variable length vector sequence can be compressed in the middle layer of the Seq2Seq model.

The encoder unit has two hidden layers: the first layer is a normal hidden layer, and the second one is a Long Short-Term Memory (LSTM) unit. The decoder unit has three hidden layers: the first and last layers are normal hidden layers, and the middle layer is an LSTM unit. We apply Dropout, the method to improve generalization performance, to the non-recurrent connection except for the input and output connection. The output of the decoder is a probability vector that represents the probability of each event appearing at the next event.

During feature extraction, a second hidden layer of the encoder when all events are inputted is extracted as a feature vector $\mathbf{f}_i$ for each of the partial event sequence $\mathbf{X'}_i$ $(0 < i \leq S \leq s)$. This operation is performed S times, and a feature vector sequence, of length S is extracted for the vector sequence $\mathbf{X}$ of the process.

3.5 Feature Classifier Training

We also use the Seq2Seq model for the feature classifier to recognize a whole feature of an event sequence from the feature sequence that is extracted from partial sequences. As mentioned in Section 3.4, a feature vector sequence of length S is extracted by the trained feature extractor from the event sequence $\mathbf{X}$ that is divided into s $(S \leq s)$. We attempt to recognize it and classify it into malicious or benign using the Seq2Seq model.

The feature classifier has almost the same structure as the feature extractor except for the dimensions of the decoder output and the application of Dropout to the output connection of the decoder. The output of the decoder is a two-dimensional vector whose elements represent the benignness and maliciousness score of a process as probabilistic values.

In the training of the feature classifier, we input the feature vector sequence $\mathbf{F} = [\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_S]$ $(S \leq s)$ to the encoder RNN in reverse order and train the feature classifier to output malicious or benign. The training is continued until the termination condition is satisfied.

3.6 Maliciousness Estimation

The process maliciousness is estimated by using the trained feature extractor and the trained feature classifier. Even in the estimation phase, as in the training phase, the vector sequence $\mathbf{X}$ of the process is divided into S partial sequences $[\mathbf{X'}_1, \mathbf{X'}_2, \ldots, \mathbf{X'}_S]$ described in Section 3.3.3, and then S $(S \leq s)$ feature vectors $\mathbf{F} = [\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_S]$ are extracted from the partial sequences by using the feature extractor. Finally, the maliciousness score of the process is estimated by obtaining the classified result from the feature vector sequence $\mathbf{F}$ using the feature classifier.

4. Experiment

To evaluate our proposal, we performed two experiments. We first evaluate the validity of the proposal by using log data recorded in a separated environment. Then we investigate if our method can detect unknown malicious processes, which means the process that is generated from malware that do not appear in training data, generated from the latest malware.

4.1 Experiment 1: Using Logs of Malware Binaries

In experiment 1, we evaluated the validity of the proposal using process logs collected by executing malware in a separated environment. Malicious or benign binaries were executed on a Windows 7 machine (Victim) under monitoring of Process Monitor to record the behavior of all processes executed on the Victim. Binary execution and monitoring were performed automatically by using Cuckoo Sandbox [*1], and the imitated environment of the Internet, simulated by Inetsim [*2] (which can simulate common Internet services such as HTTP, and DNS), was running on a network different from the Victim via the router. Monitoring time was 30 minutes from the start of the binary execution.

4.1.1 Collecting Process Logs

We collected process logs by recording logs when executing benign or malicious binaries. The collected benign process logs contains the logs of process that are generated from the benign binaries, and system processes when the binary was executed. We also randomly select system processes those name are duplicated because behavior of system process may be differ because of the executed binary. We presumed a process log to be malicious if it satisfied at least one of the following conditions and was extracted from process logs of malicious binaries.

(1) A process of a predetermined malicious binary (same name)
(2) A process generated from process (1)
(3) A process injected with malicious code from (1) or (2)

We used Cuckoo Sandbox to confirm whether or not the process satisfies condition (2) or (3).

We executed 320 malicious binaries, which were collected by NTT Secure Platform Laboratories from April to October 2014, and obtained 641 malicious process logs that are matched to the condition as we explained above. The types of these 320 binaries classified by Kaspersky are shown in **Table 2**. We also executed 37 benign binaries that were collected from commonly used software, an installer of the software, or from software originally installed in Windows, and obtained 1,000 benign process logs.

Table 2 Types of malicious binaries.

Type	#	Type	#
Adware	92	Trojan-Dropper	6
Downloader	86	Trojan-PSW	4
Trojan	61	Trojan-Ransom	2
Trojan-Downloader	21	DangerousObject	1
Backdoor	17	RiskTool	1
Trojan-Spy	15	RootKit	1
WebToolbar	12	Trojan-GameThief	1

[*1] https://cuckoosandbox.org/
[*2] http://www.inetsim.org/

4.1.2 Evaluation Method

The generalization performance of the method is evaluated using five-fold cross validation, and its efficiency is evaluated by calculating the Area Under the Curve (AUC), which is calculated from the Receiver Operating Characteristic (ROC) curve, and comparing it with AUCs of other methods. The ROC curve is a graph that shows the relationship between the True Positive Rate (TPR) and False Positive Rate (FPR) when the threshold value is changed. In our proposal, processes are estimated to be malicious (P: Positive) or benign (N: Negative) on the basis of the relationship between the maliciousness score of a process and a threshold value determined in advance. The relationship between True Positive (TP) and False Positive (FP) is shown in **Table 3**, and TPR=TP/P, FPR=FP/N.

To evaluate the efficiency of our method, we compared it with a comparison method often used by existing malware detection methods. In the comparison method, a uni-gram feature is extracted from an event sequence of a process and is then classified by using an SVM.

We used a uni-gram feature for the comparison method because it is one of the simplest but most widely used features of sequential data. A uni-gram feature is extracted in two ways in this experiment: Bag of Words (BoW) and Term Frequency (TF). In either way, an event set $\{w_1, w_2, \ldots, w_n\}$ that appears in training data is first created. Then, in BoW, a uni-gram feature is created considering only the presence/absence of each event in an event sequence. In TF, it is created considering the frequency of each event. When $b(w)$ outputs the presence/absence of the event w in an event sequence as a binary value and $f(w)$ outputs the frequency of w, the feature vector of the event sequence $\mathbf{X}$ becomes $[b(w_1), b(w_2), \ldots, b(w_n)]$ in BoW, and $[f(w_1), f(w_2), \ldots, f(w_n)]$ in TF.

We adopted an SVM for the feature classifier of the comparison method because it is one of the commonest supervised machine learning methods used for classification. The SVM classifies data using a hyper-plane that best separates the labeled data. The hyper-plane is determined by maximizing the distance from the hyper-plane to the nearest training sample. The SVM requires the hyper-parameters C, γ and kernel function. The parameter C is called a cost parameter, which determines the tolerance of misclassification, and the larger the value of C, the more misclassification is allowed. The parameter γ is used in the Radial Basis Function (RBF) kernel, and the hyper-plane can be a more complicated shape as γ increases. We use RBF kernel for the kernel function, and set $C = 1{,}000$, $\gamma = 0.001$.

4.1.3 Parameter Configuration

Hyper-parameters of the feature extractor and the classifier used in the experiment are shown in **Table 4**. Vocabulary means the kind of event appearing in training data at each cross validation, and dimension of hidden layers means the dimension of hidden-layers of the feature extractor and the classifier. L and S mean the maximum lengths of a partial event sequence and a feature sequence, respectively. An epoch means that all event sequences in training data are entered just once to the model.

Table 3 Relationship between TP and FP.

	Classified Class	
True Class	**Malicious**	**Benign**
Malicious (P)	True Positive (TP)	False Negative (FN)
Benign (N)	False Positive (FP)	True Negative (TN)

4.1.4 Results

We calculated average ROC curves using a total of 1,641 process logs of 641 malicious processes and 1,000 benign processes that were collected by the methods mentioned in Section 4.1.1. **Figure 3** shows ROC curves of each method and parameters when FPR (the horizontal axis) is a logarithmic scale. Since ROC curves are calculated for each validation, we used the average ROC curve of five calculated ROC curves for evaluation. In Fig. 3, S2S means Seq2Seq, and X-Y-Z such as 150-150-100 in the legend is the hyper-parameters. X, Y, Z mean the dimension of hidden layers, the maximum length of the input sequence, the maximum length of the feature sequence, respectively. As shown in Fig. 3, higher AUCs were obtained in the proposal when the parameters were 150-150-100 or 150-250-60. Its highest AUC was 0.979, which is 0.005 and 0.043 higher than the highest AUCs of BoW and TF, respectively. Figure 3 also shows that our proposal can estimate malicious processes with an 80% or higher TPR even if the FPR is 1%. An average of 9.2 malware families are contained in each validation, and malicious processes derived from an average of 5.8 malware families were detected under the above condition. Focusing on the hyper-parameters in the proposal, the AUCs are slightly different.

We implemented our proposal and comparison methods with Python3 and trained them on a machine with Intel(R) Core(TM) i7-4790 CPU, and NVIDIA GTX 960 with 4 GB MEM. In our proposal, training took about 10 hours and estimation took 30 minutes for each validation. Since about 280 process logs

Table 4 Hyper-parameters of feature extractor and classifier.

Parameters	**Values**
Vocabulary	11,584–12,499
Dim. of hidden layers	150
Minibatch size	35
Gradient Clipping	5
Optimizer	Adam
Termination condition	Feature Extractor: 15 epochs Feature Classifier: 25 epochs
(L, S)	$(60, 250)$ $(150, 100)$ $(250, 60)$

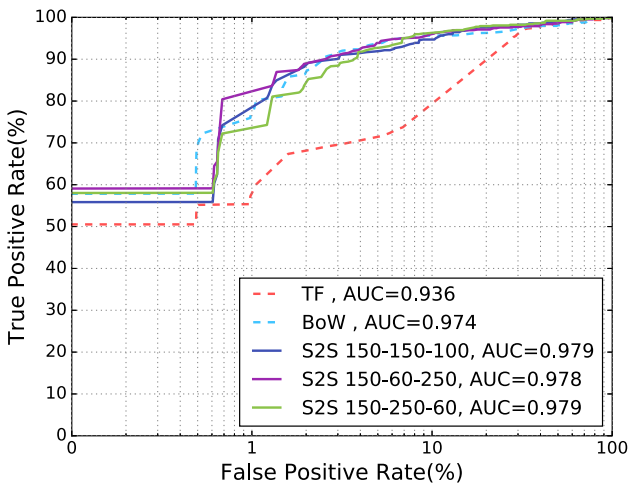


Fig. 3 ROC curves.

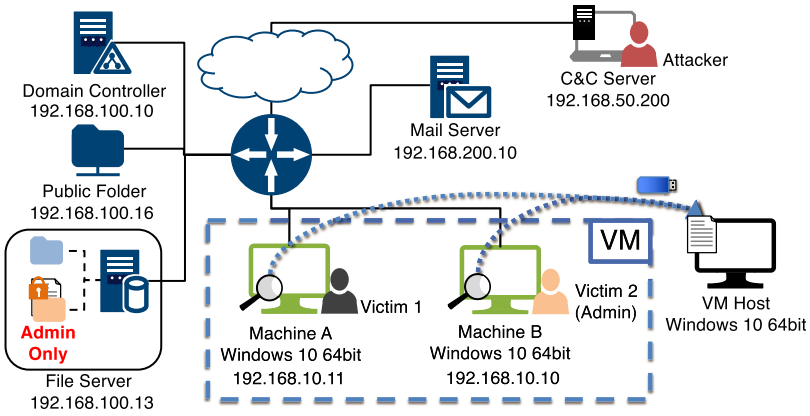


Fig. 4 Recording environment (Experiment 2).

are used for estimation in each validation, estimation speed is 6 sec./process in our method. In the comparison methods, training took 15 seconds and estimation took 1 second.

4.2 Experiment 2: Using Logs of a Simulated Attack

In experiment 2, we investigated whether our method can detect an unknown malicious process, which mean the process that is generated from malware which does not appear in training data, by performing a simulated attack in a more realistic environment.

4.2.1 Recording Environment

The recording environment for experiment 2 is shown in **Fig. 4**. This environment consisted of a Domain Controller, Mail Server, two File Servers, and two client machines. Machine A is operated by Victim 1 and the privileged Machine B is operated by administrator Victim 2. These clients are constructed in the virtual environment to simplify rollback. We manage user accounts and file permissions of each client with an Active Directory server as the Domain Controller. The Mail Server manages email accounts of Victims 1 and 2. Almost all files and folders in the File Server can be accessed by any clients, but some folders require the administrator's authorization to access.

We deployed the C&C Server, which is operated by the Attacker, in this environment to execute a simulated attack and performed the attack under monitoring to record the behavior of malicious processes executed on Machine A and/or B.

4.2.2 Simulated Attack Scenarios

We simulated an attack to record realistic logs using some Remote Access Tool (RAT) simulators and post exploitation frameworks: ShinoBOT SUITE [*3], Koadic [*4], and PowerShell-Empire [*5] (PSE). We built seven attack scenarios that follow each stage of targeted attacks step by step and performed attacks against the recording environment in accordance with each scenario.

Scenario 1: Initial infiltration

Scenario 1 simulates the initial infiltration phase. In this scenario, an attacker sends a target email with the ShinoBOT downloader attached to Victim 1 and infiltrates into the target network by infecting Machine A used by Victim 1 with ShinoBOT.

Scenario 2: Building an attack infrastructure

Scenario 2 simulates building an attack infrastructure phase. The attacker first collects system information of Machine A and the network configuration by using ShinoBOT and then installs Koadic. The attacker escalates privilege using `sdclt.exe` and steals authentication information using mimikatz [*6] and MailPassView [*7].

Scenario 3: Building an attack infrastructure via HTTP

Scenario 3 is almost the same as scenario 2, but the attacker steals email authentication information by uploading files that contain the information via HTTP instead of using MailPassView.

Scenario 4: Probing the system

In scenario 4, the attacker deploys a PSE downloader imitating a document file to the Public Folder. Then Victim 2 accidentally executes the downloader and Machine B used by Victim 2 is infected with PSE.

Scenario 5: Pursuing the final goal of the attack

Scenario 5 simulates the purpose accomplishment phase. In this scenario, the attacker uploads stolen confidential files stored in the File Server to C&C server by using `ftp` command. The attacker first compresses the file at Machine B, moves the compressed file to Machine A, and then uploads the file from Machine A.

Scenario 6: Pursuing the final goal of the attack via HTTP

Scenario 6 also simulates the purpose accomplishment phase. The difference from scenario 5 is the attacker uploads files via HTTP instead of `ftp` and only uses Machine B.

Scenario 7: Preparation of re-intrusion

In Scenario 7, the attacker erases the traces of intrusion and prepares re-intrusion. First, the attacker replaces a desktop shortcut with a script for re-installing RATs. Then the attacker erases event logs, downloaded files and installed RATs from machines. We use SDelete [*8] for deleting files.

4.2.3 Collecting Process Logs

The scenario was executed in the order of 1 → {2 or 3} → 4 → {5 or 6} → 7, and we collected 490 malicious process logs. **Ta-**

[*3] https://shinosec.com/shinobotsuite/
[*4] https://github.com/zerosum0x0/koadic
[*5] https://github.com/EmpireProject/Empire
[*6] https://github.com/gentilkiwi/mimikatz
[*7] https://www.nirsoft.net/utils/mailpv.html
[*8] https://technet.microsoft.com/ja-jp/sysinternals/sdelete.aspx

Table 5 Number of malicious process logs extracted from each scenario.

Scenario (Machine)	# of malicious process
Scenario 1 (A)	48
Scenario 2 (A)	228
Scenario 3 (A)	35
Scenario 4 (A)	49
Scenario 4 (B)	8
Scenario 5 (A)	53
Scenario 5 (B)	4
Scenario 6 (B)	7
Scenario 7 (A)	30
Scenario 7 (B)	28
Sum	490

ble 5 shows the number of extracted malicious process logs for each scenario.

We collected malicious process logs from Machine A and/or B when executing a simulated attack in accordance with scenarios 1–7. We collected benign process logs for 30 minutes from a client PC running in the recording environment in a normal state, and a Windows PC running in the real environment. We collected 117 benign process logs from a client PC in the normal state and 178 benign process logs from the PC running in the real environment, which is connected to the Internet and have some client machines on the same network.

Then we created a dataset from the collected 785 process logs and 1,641 process logs used in experiment 1.

Training Data Training data was the 1,641 process logs used in experiment 1, 30 malicious process logs of scenario 7 (A), and 178 benign logs of the Windows machine running on the real environment.

Test Data Test data was the remaining 577 process logs.

4.2.4 Evaluation Method

In this experiment, we use *F-measure* and *Specificity* showed in Eqs. (3) and (4) for evaluation because the test data of each scenario was too small to calculate ROC curves.

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

$$F\text{-}measure = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{3}$$

$$Specificity = \frac{TN}{TN + FP} \tag{4}$$

F-measure is a harmonic average of precision and recall values, and *Specificity* is a percentage of what is expected to be benign out of what is actually benign. Therefore, the malicious process detectability is high in the method in which both values are higher.

We set the parameters of the proposal almost the same as those of experiment 1, but L and S were fixed to 150 and 100. To show the validity of the results, we compared the results of our proposal with those of the comparison method (BoW) explained in Section 4.1.2.

4.2.5 Results

We calculated *F-measure* values of our proposal and BoW for each scenario and *Specificity* value using the test dataset mentioned in Section 4.2.3. **Figure 5** shows the values of *F-measure* calculated from the all benign process logs of test data and the malicious process logs of each scenario and shows the value of

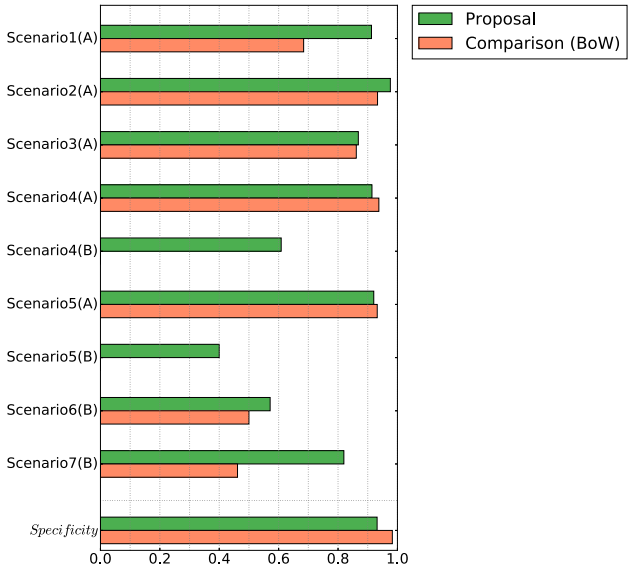


Fig. 5 *F-measure* values for each scenario and *Specificity* value.

Specificity. Since the classifier trains the maliciousness score as a probability value, we set 0.5 (50%), which is the intermediate value, as a threshold value.

As shown in the graph of each scenario in Fig. 5, our proposal obtained higher *F-measure* values than the comparison method in many scenarios. Particularly for Machine A in scenario 1 and Machine B in scenarios 4, 5, 6, and 7, which is a scenario using a different RAT simulator from learning data, our proposal showed F-measure values especially higher than the comparison method. Besides, the *Specificity* value in Fig. 5 shows that our proposal correctly classified 90% or more benign processes.

5. Discussion

5.1 Discussion of Experimental Results

From experiment 1, our proposal properly estimates malicious processes even if the process is derived from different malware families, therefore, our proposal can potentially detect different kinds of malware. The result of experiment 1 also showed that our proposal needs 6 seconds for estimating a process. Our proposal record process logs for X seconds (X is 1,800 in the experiment), and then performs the estimation. Therefore, our proposal has practical speed for estimation if all recorded process logs are estimated malicious or benign in X minutes. An elapsed time for estimation is calculated by $num\ of\ process \times 6$. Assuming that $X = 1{,}800$, we can estimate the number of recorded process may be 100 to 200 from the Section 4.2.3 and estimated elapsed time is 600 to 1,200 seconds. Thus, our proposal can estimate processes in practical speed.

The result of experiment 2 shows that our proposal works against recent malware even if the amount of training malicious logs derived from recent malware is small. This may be because recent malware has a similar behavioral feature to the old malware, so our proposal learns the generic feature from old malware and can detect recent malware.

5.2 Extracted Feature Analysis

We adopted the Seq2Seq model to our proposal to compress in-

Table 6 Example of event sequence prediction using trained feature extractor.

Order	Inputs	Predicted Events
1	QueryBasicInformationFile	QueryBasicInformationFile
2	C:\Users\sun\<omit>\	C:\Users\sun\<omit>\
3	SUCCESS	SUCCESS
4	CloseFile	CloseFile
5	C:\Users\sun\<omit>\	C:\Users\sun\<omit>\
6	SUCCESS	SUCCESS
7	CreateFile	CreateFile
8	C:	C:
9	SUCCESS	SUCCESS
10	FileSystemControl	QueryDirectory
11	C:	C:\Users
12	INVALID DEVICE REQUEST	SUCCESS
...	...	...
94	FileSystemControl	CreateFile
95	C:\Users\sun\<omit>\	C:\Users\sun\<omit>\<exe>
96	INVALID DEVICE REQUEST	SUCCESS
97	QueryDirectory	QueryAttributeTagFile
98	C:\Users\sun\<omit>\	C:\Users\sun\<omit>\<exe>
99	SUCCESS	SUCCESS
100	CloseFile	QueryBasicInformationFile
101	C:\Users\sun\<omit>\	C:\Users\sun\<omit>\<exe>
102	SUCCESS	SUCCESS
103	RegQueryKey	CloseFile
104	HKCU\Software\Classes	C:\Users\sun\<omit>\<exe>
105	SUCCESS	SUCCESS
...	...	...

formation of an input sequence and extract features considering a time-series of the input sequence. In this section, we analyze if the feature extractor actually compresses information of input sequences and learns a time-series of input sequence by predicting a sequence using a trained feature extractor.

Table 6 shows an example of a partial predicted sequence. We used the log of a benign system process in test data for input data. Order shows the order of input/predicted sequence, Inputs shows the sequence inputted to the trained feature extractor, and Predicted Events shows the sequence outputted from the extractor. Predicted Events shows the extractor correctly learns the order of events: Operation → Path → Result. It also shows a file path is predicted when the previous Operation is related to file access such as the prediction of order 94 and 95. Furthermore, the extractor probably recognizes behavior that is constructed with multiple Operations such as "`CreateFile` → `some operations related to file access` → `CloseFile`" from the prediction of order 94–105. Therefore, the feature extractor certainly seems to learn the time-series of the sequence.

Next, we discuss from the perspective of the relationship between inputs and prediction. Table 6 shows that the beginning and ending of the event sequence are predicted roughly correctly, but the predicted events of order 94–105 are wrong. Therefore, the extracted feature retains partial information of the input sequence.

5.3 Influence of Event Presence Ratio

Although the proposed method obtained a quite high precision performance of AUC=0.979 at best, the comparison method (BoW) also obtained an almost equally high performance of AUC=0.974. These results suggest that the presence/absence of events is the important factor for estimation in the proposal. In this section, we investigate the cause of misclassification in our proposal to infer the important factors for estimation. In the following, TP/FP and TN/FN sets mean sets of malicious/benign

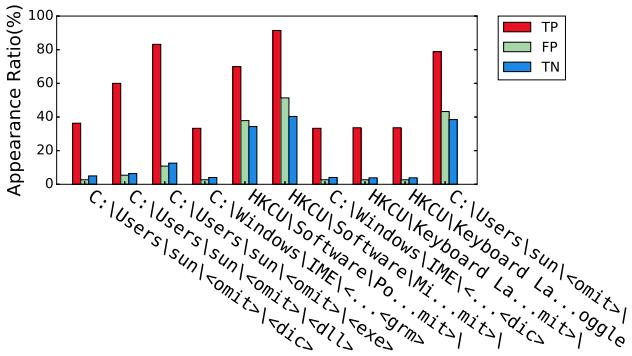


Fig. 6 Events and their presence ratios of TP, TN, and FP sets in experiment 1.

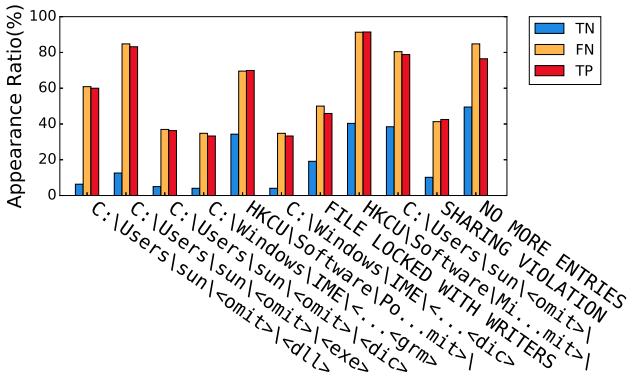


Fig. 7 Events and their presence ratios of TN, TP, and FN sets in experiment 1.

processes whose maliciousness score is more than 0.5 and less than 0.5, respectively. Presence ratio means the percentage of event sequences containing the event among event sequences of all processes belonging to a specific set. We investigated the result of the proposal whose parameter was 150-150-100, and the numbers of TP, FN, TN, and FP sets were 595, 46, 963, and 37, respectively.

Figure 6 shows the events in which the presence ratio differs by more than 30 points between the TP and FP sets or TN and FP sets, and **Fig. 7** shows the events in which the presence ratio differs by more than 30 points between the TN and FN sets or TP and FN sets. The X axis represents the events, and the Y axis represents the presence ratio of each set. For example, the leftmost bars in Fig. 6 show the event `C:\Users\sun\<omit>\<dic>` is contained in less than 10% of the event sequences of processes belonging to the TN and FP sets but about 40% of the sequences belonging to the TP set. Figures 6 and 7 show that the presence ratio distributions are similar for the TN and FP sets or TP and FN sets. The true class of TN and FP sets, or TP and FN sets are same, so that the cause of misclassification is not the presence ratio but any other features. Thus, the presence of events is not the important factor for the proposal. From these results and results in Section 5.2, the order of events is presumed to affect the estimation performance in our proposal.

5.4 Superiority and Limitations

The results of experiment 1 show our proposal classifies malicious processes at least as the same as the comparison method (BoW). The results of experiment 2 show that our proposal can

detect malicious processes derived from RAT simulators that do not appear in the training data. In addition, Sections 5.2 and 5.3 reveal that the order of events is important for our proposal. Thus, our proposal is superior to the comparison method in terms of the generalization performance and can be said to extract and classify features even from unknown malicious processes.

On the other hand, our proposal has several limitations. First, we supposed that the client machine is constantly monitored and data is gathered with short intervals. However, this assumption may affect the performance of the machine. Second, our proposal cannot show the reason for the result obtained by classification. Some techniques to visualize effective features such as attention mechanisms [21] have been proposed, but further study will be required to apply these methods to our proposal, because our model applies Seq2Seq model stepwise. Third, our proposal depends on the logs generated from ProcessMonitor. This software only works on the Windows OS, thus our proposal is only available for Windows machines. Moreover, if malware that cannot be recorded by ProcessMonitor such as Bootkit, or which is just running and does nothing are used, our proposal cannot detect them. Our proposal is based on the idea that the behavior of malicious process and benign process is different, so the maliciousness of a process that has very similar behavior to the malicious process may become high.

6. Conclusion

In this paper, we proposed a method to estimate process maliciousness by extracting features from the behavior of processes running on a machine and classifying them into malicious or benign. This method effectively compresses event sequences into feature vector by applying Seq2Seq models stepwise. We evaluated our proposal in two experiments and found the following. First, we showed our proposal obtains a higher AUC value (0.979) than the comparison methods using uni-gram feature by performing 5-fold cross validation using 1,641 process log recorded by Process Monitor. Second, we revealed the order of events may be the important factor for estimation in our proposal by analyzing the extracted features and event presence ratio of the classified results. Finally, we indicated our proposal can detect malicious processes derived from a recent malware simulator that do not appear in the training data.

Our future work is to devise a new way to show reasons for estimation. Moreover, we might collect many more malicious process logs and more various benign binaries to reduce FPR and increase TPR. We should also compare the proposal and commercial products.

References

[1] Kobayashi, H., Konno, C., Souma, M., Ohmori, M. and Irisawa, Y.: Design and Operational Guide to Protect against "Advanced Persistent Threats" Revised 2nd edition, Information-tehnology Promotion Agency (online), available from ⟨https://www.ipa.go.jp/files/000017299.pdf⟩ (accessed 2018-02-16).

[2] Hutchins, E.M., Cloppert, M.J. and Amin, R.M.: Intelligence-Driven Computer Network Defense Informed by Analysis of Adversary Campaigns and Intrusion Kill Chains, *Proc. 6th Int. Conf. Information Warfare and Security*, pp.113–125 (2010).

[3] Hatta, J., Ishikawa, Y. and Kaneko, H.: Cyber GRID View Vol.1 Research Report on Advanced Persistent Threats in Japan, Lac Corporation (online), available from ⟨https://www.lac.co.jp/english/report/pdf/apt_report_vol1_en.pdf⟩ (accessed 2018-02-16).

[4] Nakamura, Y.: ChChes - Malware that Communicates with C&C Servers Using Cookie Headers, Japan Computer Emergency Response Team Cordination Center (online), available from ⟨http://blog.jpcert.or.jp/2017/02/chches-malware--93d6.html⟩ (accessed 2018-02-16).

[5] Japan Computer Emergency Response Team Cordination Center: Detecting Lateral Movement through Tracking Event Logs, Japan Computer Emergency Response Team Cordination Center (online), available from ⟨https://www.jpcert.or.jp/english/pub/sr/20170612ac-ir_research_en.pdf⟩ (accessed 2018-02-16).

[6] Ahmed, F., Hameed, H., Shafiq, M.Z. and Farooq, M.: Using Spatio-Temporal Information in API Calls with Machine Learning Algorithms for Malware Detection, *Proc. 2nd ACM Workshop on Security and Artificial Intelligence*, pp.55–62 (2009).

[7] Ravi, C. and Manoharan, R.: Malware Detection using Windows API Sequence and Machine Learning, *Int. J. Computer Applications*, Vol.43, No.17, pp.12–16 (2012).

[8] Tian, R., Islam, R., Batten, L. and Versteeg, S.: Differentiating malware from cleanware using behavioural analysis, *Proc. 5th Int. Conf. Malicious and Unwanted Software*, pp.23–30 (2010).

[9] Wang, W., Zhu, M., Zeng, X., Ye, X. and Sheng, Y.: Malware traffic classification using convolutional neural network for representation learning, *Proc. 31th Int. Conf. Information Networking*, pp.712–717 (2017).

[10] Saxe, J. and Berlin, K.: Deep Neural Network Based Malware Detection Using Two Dimensional Binary Program Features, *Proc. 10th Int. Conf. Malicious and Unwanted Software*, pp.11–20 (2015).

[11] Pascanu, R., Stokes, J.W., Sanossian, H., Marinescu, M. and Thomas, A.: Malware classification with recurrent networks, *Proc. 40th IEEE Int. Conf. Acoustics, Speech and Signal Processing*, pp.1916–1920 (2015).

[12] Wang, X. and Yiu, S.M.: A Multi-task Learning Model for Malware Classification with Useful File Access Pattern from API Call Sequence, *CoRR*, Vol.abs/1610.05945 (2016).

[13] Hardy, W., Chen, L., Hou, S., Ye, Y. and Li, X.: DL4MD: A Deep Learning Framework for Intelligent Malware Detection, *Proc. 12th Int. Conf. Data Mining*, pp.61–67 (2016).

[14] Nakazato, J., Tsuda, Y. and Takagi, Y.: A Suspicious Process Analysis in Cooperation with End Hosts, Technical Report 2, National Institute of Information and Communications Technology (2017).

[15] Symantec: About SONAR, Symantec (online), available from ⟨https://support.symantec.com/en_US/article.HOWTO80968.html⟩ (accessed 2019-03-26).

[16] Kaspersky: Behavior-based Protection, Kaspersky (online), available from ⟨https://www.kaspersky.com/enterprise-security/wiki-section/products/behavior-based-protection⟩ (accessed 2019-03-26).

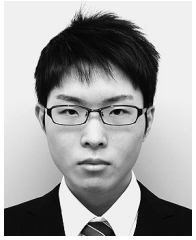
[17] FFRI, Inc.: Targeted Attack Protection Software FFRI yarai, FFRI Inc. (online), available from ⟨https://www.ffri.jp/assets/files/products/ctrg/yarai_E_brochure.pdf⟩ (accessed 2019-03-26).

[18] Tobiyama, S., Yamaguchi, Y., Shimada, H., Ikuse, T. and Yagi, T.: Malware Detection with Deep Neural Network Using Process Behavior, *Proc. 6th IEEE Int. Workshop on Network Technologies for Security, Administration and Protection*, pp.577–582 (2016).

[19] Sutskever, I., Vinyals, O. and Le, Q.V.: Sequence to Sequence Learning with Neural Networks, *Proc. 27th Int. Conf. Neural Information Processing Systems*, Vol.2, pp.3104–3112 (2014).

[20] Russinovich, M. and Cogswell, B.: Process Monitor, TechNet-Microsoft (online), available from ⟨https://technet.microsoft.com/ja-jp/sysinternals/processmonitor.aspx⟩ (accessed 2018-02-16).

[21] Bahdanau, D., Cho, K. and Bengio, Y.: Neural Machine Translation by Jointly Learning to Align and Translate, *CoRR*, Vol.abs/1409.0473 (2014).



Shun Tobiyama received his M.E. degree in Information Science from Nagoya University, Japan in 2018. He joined Nippon Telegraph and Telephone Corporation (NTT) in 2018, and is now with the Secure Architecture Project of NTT Secure Platform Laboratories. His research interest is malware detection techniques, network security, and machine learning.

Yukiko Yamaguchi graduated in 1983 from the Department of Information Engineering, Nagoya Institute of Technology, and obtained her Master's degree in 1982 from Nagoya University. She then affiliated with Fujitsu Laboratories Ltd. In April 1991 she joined Nagoya University as an Assistant Professor in the Computer Center. At present, she is an Assistant Professor in Information Technology Center and engaged in research on network management technology and cyber security. She is a member of IPSJ and IEICE.

Hirokazu Hasegawa received his Ph.D. degree in Information Science from Nagoya University, Japan, in 2017. He is currently an Assistant Professor at Information Strategy Office, Nagoya University, Japan. His research interests include the Internet and network security. He is a member of IPSJ and IEICE.

Hajime Shimada received his B.E., M.E. and D.E. degrees from Nagoya University, Japan in 1998, 2000 and 2004 respectively. He was an assistant professor in Kyoto University from 2005 to 2009. He wan an associate professor in NAIST from 2009 to 2013. He is now an associate professor in Nagoya University, Japan since 2013. He is currently focusing on both power efficient computer architectures and cyber security techniques including network security and malware detection. He is a member of IEEE, IPSJ, 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 Nippon Telegraph and Telephone Corporation (NTT) in 2007, he has been engaged in research and development on cybersecurity. He is currently a Senior Distinguished Researcher with the Cyber Security Project of NTT Secure Platform Laboratories. His research interests include cybersecurity measurement, offensive security, and usable security and privacy.

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 and transferred to NTT Security (Japan) KK in 2018, where he is currently researching honeypots, security-data analysis based on machine learning, and security intelligence technologies such as URL/domain/IP blacklisting and reputation. He is a member of the Institute of Electrical and Electronics Engineers (IEEE) and the Institute of Electrical Engineers of Japan (IEEJ) and IEICE.