Electronic Preprint for Journal of Information Processing Vol.28

Regular Paper

Detection of Malicious Tools by Monitoring DLL Using Deep Learning

Wataru Matsuda[1,†1,a)] Mariko Fujimoto[1,†2,b)] Takuho Mitsunaga[2,c)]

Received: April 10, 2020, Accepted: September 10, 2020

Abstract: In targeted attacks, various malicious tools are leveraged by attackers. According to the Cybersecurity and Infrastructure Security Agency (CISA), tools such as China Chopper, Mimikatz, PowerShell Empire, and HUC Packet Transmitter are used in targeted attacks. Standard malware detection methods include those based on file names or hashes. However, attackers tend to avoid detection by changing the file name of malicious tools or by rebuilding them. Therefore, detecting malicious tools used in targeted attacks is difficult. We found that the order of Windows built-in DLLs loaded by each malicious tool has unique characteristics. In this study, we propose a detection method of malicious tools by analyzing DLL information using deep learning, considering the DLL and its order of loading by each process. We confirmed that even if the file names are changed or tools are rebuilt, our proposed method could detect the mentioned four tools with high detection rates: with a recall rate of 97.45%, a precision rate of 97.29%, and F value of 97.37% on average. Furthermore, the proposed method can detect malicious tools with more than a 90% detection rate, even if about 10% of loaded DLLs are changed in the future.

Keywords: deep learning, long short-term memory, DLL, sysmon, targeted attacks, detection

1. Introduction

Targeted attacks have become a serious threat to computer systems. In targeted attacks, various tools are leveraged by attackers. However, detecting targeted attacks is challenging because the methods of the attacks are sophisticated. According to Ref. [1], the following five tools are seen in cyber incidents worldwide.

- JBiFrost: A kind of remote access trojan. Once installed on a victim's computer, it allows remote administrative control.
- China Chopper: A webshell that has been in widespread use since 2012.
- Mimikatz: Mainly used by attackers to collect the credentials of other users who are logged into a targeted Windows machine.
- PowerShell Empire: Designed for lateral movement after gaining initial access.
- HUC Packet Transmitter: A proxy tool used to intercept and redirect Transmission Control Protocol (TCP) connections in order to obfuscate attackers'communications with victim networks.

Recently, detection utilizing Endpoint Detection and Response (EDR) [*1] on each computer is becoming more common [2]. As part of EDR, malware detection based on file name or hash is one of the more popular methods [3]. However, attackers tend to avoid detection by changing the file names of malicious tools or by rebuilding them. Therefore, detecting these malicious tools is considered difficult [4], [5], [6]. To solve this problem, previous research introduced detection methods using DLLs and analyzing suspicious files [7], [8], [9]. DLL (Dynamic Link Library) is a shared library used by Windows processes, and some malicious tools also use legitimate DLLs. If legitimate DLLs loaded by malicious tools have unique characteristics, it is possible to detect malicious tools even if the file names of the tools have changed or the source code of the tool is rebuilt. These previous researches [7], [8], [9] are useful solutions for analyzing files that are already identified as malicious in an isolated environment. However they do not consider their use in the running environment, thus these methods need forensic tools and are supposed to analyze in an isolated environment such as a sandbox environment. This renders these methods not necessarily suitable for detection in the production environment.

This research focuses on the detection method in the running environment (computers used in daily works). For the detection method using DLL information in the running environment, a detection method of monitoring DLLs that are commonly loaded by malicious tools using DLL lists [10] has been previously researched. Although, the DLLs loaded by malicious tools are different depending on the version of Windows and the malicious tools. The method lacks flexibility since if the loaded DLLs are changed, it would not be able to detect the malicious tool.

We found the unique characteristics of DLLs and their order loaded by each malicious tool. In this study, we propose a flexible detection method of malicious tools by analyzing DLL infor-

[1] The University of Tokyo, Bunkyo, Tokyo 113–8654, Japan
[2] Toyo University, Kita, Tokyo 115–0053, jJapan
[†1] Presently with NTT Secure Platform Laboratories
[†2] Presently with NEC Solution Innovator
[a)] wataru.matsuda.ev@hco.ntt.co.jp
[b)] marikof@nec.com
[c)] takuho.mitsunaga@iniad.org

[*1] Emerging technology that focuses on identifying and exploring malicious activities on endpoints.

mation using deep learning techniques. We examine four famous malicious tools: China Chopper, Mimikatz, PowerShell Empire, and HUC Packet Transmitter as the dataset, and evaluate the effectiveness of the proposed method. As a result, we confirmed that the proposed method could detect these four tools with high accuracy, even if some of the loaded DLLs are changed.

The following are the contributions of this proposed method.

- Even if DLLs that are loaded by malicious tools are changed in the future, our proposed method would detect them. We were able to confirm that this proposed method is capable of detecting malicious tools with a recall rate and precision rate of more than 90%, even when about 10% of the loaded DLLs are changed.
- Even if attackers change the malicious tools' filenames or rebuild the tools, our method can detect them with high accuracy by analyzing the characteristics of legitimate Windows DLLs. It is because of that if attackers modify the legitimate Windows DLLs, the system would not work correctly.
- Our method can minimize the affect system performance. Therefore the method is suitable for detection in the running environment.
- Our method can detect malicious tools just by their startup. Therefore it is useful for immediate incident response.

In Section 2, we will describe the characteristics of targeted attacks and the difficulty in the detection of malicious tools. Section 3 describes related research. Section 4 describes the detail of the proposed method. Section 5 describes the evaluation methods of the proposed methods and their results.

2. Targeted Attacks Using the Malicious Tools

2.1 Malicious Tools Used in Targeted Attacks

Attackers of a targeted attack have a clear objective, such as stealing a specific organization's information. Attackers who can intrude into the organization tend to expand infections until they accomplish their goal. Although it is difficult to prevent intrusions, it is possible to minimize the damages if it is possible to detect attack activities in the early stage. According to Ref. [1], the following five tools are widely used in targeted attacks.

- JBiFrost: A cross-platform and multifunctional remote access trojan. It poses a threat to several different operating systems, including Windows, Linux, MAC OS X and Android. Once installed on a victim's computer, it allows remote administrative control.
- China Chopper: A web based remote shell tool that has been in widespread use since 2012.
- Mimikatz: Widely used Windows hacking tool. it is mainly used by attackers to collect the credentials of users, creating malicious authentication tickets, remote access leveraging legitimate credentials, etc.
- PowerShell Empire: A pure PowerShell post-exploitation agent. Empire implements the ability to run PowerShell agents without needing powershell.exe, rapidly deployable post-exploitation modules ranging from key loggers to Mimikatz, and adaptable communications to evade network detection, all wrapped up in a usability-focused framework.
- HUC Packet Transmitter: A proxy tool used to intercept and redirect Transmission Control Protocol (TCP) connections in order to obfuscate attackers' communications with victim networks.

Table 1 The number of DLLs of each malicious tool.

Tool name	common DLLs	All Dlls
China Chopper	23	110
Mimikatz	20	63
PowerShell Empire	49	135
HUC Packet Transmitter	8	42

2.2 Difficulty in Detecting Malicious Tools

It is difficult to detect malicious tools used for targeted attacks. Usually, attackers try to avoid detection, by such methods as changing the file name of the malicious tools, rebuilding them, and so forth. For example, since the source code of Mimikatz is published on the internet, attackers can rebuild it to create their own tools. According to the result of scanning services such as VirusTotal [11] detection rate is poor after rebuilding [5], [12]. The detection rate of Mimikatz without modification was 100%. After the trivial modification of replacing the word "Mimikatz" with other words such as "kitikatz," the detection rate was only 7.2% (4 out of 54) [12]. It is clear that for security products that use the file name or hash value for detection, detection is difficult if attackers customize malicious tools.

2.3 Detection Using DLL Loaded by Malicious Tools

To solve the problem shown in Section 2.2, some researchers have introduced detection methods using DLLs loaded by malicious tools [7], [8], [10], [13]. If legitimate Windows DLLs loaded by malicious tools have unique characteristics, the method could detect the tools even if the file names are changed or the tools themselves are rebuilt. According to previous research [10], DLLs loaded by malicious tools are dependent on the version of Windows and the malicious tools. We investigated unique DLLs loaded by malicious tools in the evaluation environments (several malicious tools versions * 9 Windows version) based on the research [10]. **Table 1** shows the number of DLLs that are loaded by each malicious tool. According to the result, the number of DLLs that were loaded regardless of the environment (common DLLs) is quite few. For instance, mimikatz loads 63 unique DLLs in total. On the other hand, there were only 20 common DLLs that were loaded regardless of the mimikatz and Windows version. These results indicate that a scalable method is necessary since DLLs loaded by malicious tools could change in future updates.

3. Previous Research

Several studies have proposed methods of detecting attacks using DLL information. In addition, there are several studies that have focused on detecting malware using Windows API calls. In this section, we introduce related research that uses DLL information and Windows API call for detection. **Table 2** shows the outline of the previous research. "Detection performance" in Table 2 shows the best detection performance of each research.

Table 2 Previous research.

Environment	Research	Analysis target	Method	Detection performance
Running environment	[19]	System call	API Hooking	Stop Unknown Malicious Code, attacks and intrusion through heap overflow
	[20]	System call	Dependency Structure Matrix	Accuracy 95%
	[21]	DLL	Support Vector Machine(SVN) Binary Decision Tree Discriminant Analysis	Accuracy 88.45%
	[10]	DLL	Signature-based (DLL list)	4 tools:100% accuracy, 1 tool:FP 0.55%
	proposed method	DLL	Long short-term memory(LSTM)	Accuracy 99.82%
Sandbox	[7]	DLL	Random Forest	Accuracy 95.02%
	[8]	DLL	SVM	-
	[14]	System call	Pattern matching of certain functions or API call sequence	Accuracy 99.89%
	[9]	System call	API hooking and DLL injection	-
	[15]	System call	10 machine learning algorithm such as Resilient Backpropagation, Levenberg-Marquardt ,etc.	The best average accuracy 89.25%
	[13]	DLL	Random Forest	100% DR with only 0.3% FP rate (It depends on the dataset)
	[16]	DLL, System call	AdaBoost, Gradient Classifier, Logistic Regression	Accuracy 94.64%
	[17]	System call	SVM,IBL,Decision Tree	Precision 78.38% average where 3 types of machine learning were combined
	[18]	DLL	DBSCAN	Correctness 94.12%

3.1 Detection in the Sandbox Analysis Environment

This section describes the analysis or classifying method of malware for computer forensics purposes. These methods are use in the sandbox analysis environment.

Several works analyze DLLs using the Cuckoo Sandbox [7], [8]. The Cuckoo Sandbox is a solution for analyzing files that are already identified as malicious in an isolated environment. Ki et al. [14] and Willems [9] introduce malware analysis methods using Windows API call sequences. This research, similar to that utilizing the Cuckoo Sandbox, focused on closely analyzing programs that had already been identified as malicious. Gonzalez et al. introduce a malware classification method using information regarding the number of Windows API calls made from a DLL [15]. Narouei et al. propose a malware detection method based on static analysis that extracted behavioral features from a unique structure in portable executables called the “DLL dependency tree” [13]. Ijaz et al. evaluated several existing static and dynamic malware analysis methods using several features including API calls and DLLs with machine learning [16]. Reference [17] presents a method to classify malware codes in static analysis using several features including Windows API calls. Reference [18] proposes a malware classifying method using DLL information in memory forensic.

From the above research, it can be said that DLLs and Windows API called from processes are useful to detect or classify malicious tools.

3.2 Detection in the Running Environment

This section describes the detection method of previous research in the running environment. These researches focus on the detection on computers used in daily works.

Sun et al. propose a method for detecting malicious shellcodes injection by monitoring Windows API calls [19]. Reference [20] proposes the detection method of persistence activities by malware such as manipulation of the registry through a monitoring system call. References [19] and [20] focus on detecting attack activities by malwares.

On the other hand, the following research focuses on DLL itself loaded by processes. Gong et al. propose a malware classification method that analyzed the DLLs loaded by the program with the Bag of Words model. The order in which the DLLs are loaded is not taken into consideration [21]. Matsuda et al. propose a detection method of startup of malicious tools by monitoring DLLs that are commonly loaded by malicious tools using the DLL lists [10].

3.3 Issues of Previous Research

This section describes issues of previous research.

3.3.1 Detection in the Sandbox Analysis Environment

These researches are useful solutions for analyzing files that are already identified as malicious in an isolated environment. However they do not consider use in the running environment, thus these methods need forensic tools such as IDA Pro, Dependency Walker etc. [13], [15]. In addition, they are supposed to analyze in the isolated environment such as the sandbox environment [7], [8], [9]. This renders these methods not necessarily suitable for detection in the production environment.

3.3.2 Detection in the Running Environment

Reference [19] monitors Windows API calls, it affects system performance. Authors have reported that their methods experience an 8%–9% performance drop.

Reference [20] detects malware’s persistence activities such as modifying the registry key. Persistence activities mean attackers have already gotten complete administrative privileges on the computer, thus more immediate detection is desirable.

Reference [10] proposed a detection method of startup of malicious tools. This research uses the “common DLL list” [22] to monitor DLLs that are commonly loaded by malicious tools regardless of their execution environment. This method lacks flexibility since loaded DLLs depend on which version the malicious tools are. This means that if one of the DLLs in the DLL list loaded by a certain malicious tool is removed in a future update, the method would not be able to correctly detect the software. We refer to this research and expand upon this detection method to be more flexible.

4. Proposed Method

We propose a method that detects malicious tools by analyzing DLL information using deep learning.

The novelty of our method is that we have analyzed both the name of DLLs and the order of DLL loading using deep learning and have presented a flexible detection method that is suitable for the production environment. To evaluate the contribution of the order of DLL loading, we also evaluate the performance of the method which does not consider the DLL order consideration in Section 4.6. We examine four malicious tools used in targeted attacks: Mimikatz, China Chopper, PowerShell Empire, and HUC Packet Transmitter to evaluate the effectiveness of our method. Note that JBiFrost is excluded from our research since the tool requires the payment of a subscriber fee [23]. We should not contribute to attackers.

4.1 Summary of the Proposed Method

Figure 1 shows an overview of the proposed method. Our proposed method consists of a learning phase and a detection phase.

- Learning phase: The dataset is created by simultaneously running malicious tools and normal processes on Windows computers and using Sysmon [24] to extract information about loaded DLLs. Information about the order in which each DLL is loaded is also monitored and added to the dataset. The dataset is then analyzed using deep learning and a classification model is created.
- Detection phase: Collect information on DLLs loaded by each process on the production environment using Sysmon. The classification model created in the Learning phase is used to identify each process as a "China Chopper", "Mimikatz", "PowerShell Empire", "HUC Packet Transmitter", or a normal process.

The details of each technique are explained in sections shown in **Table 3**.

4.2 Solution for Issues of Previous Research

This section describes how to solve the issues of previous research in the proposed method.

- Our method only uses Sysmon in the running environment, and analyses Sysmon logs exported from the running environment. Thus it can minimize the effect on system performance (only A few percent increase in memory and disk usage). Therefore the method is suitable for detection in the running environment.
- Our method detects malicious tools just by their startup before malicious activities such as memory dump or persistence activities are conducted. Therefore it is useful for immediate incident response.
- Our method provides a scalable detection method by monitoring the order of DLL loading loaded by malicious tools using deep learning. Our method solves the problem in previous research such as Ref. [10] that malicious tools are overlooked if the use of any of the DLLs in the common DLL list is discontinued.

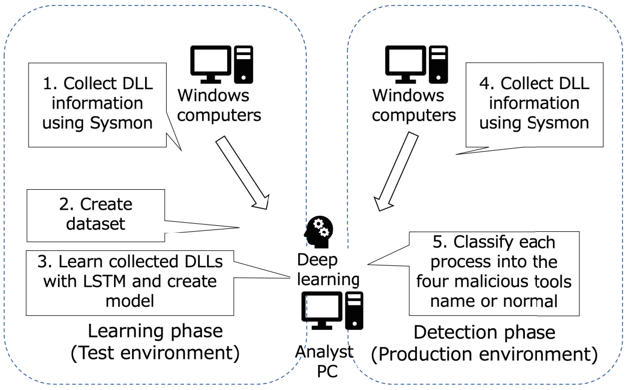

1. Collect DLL information using Sysmon
Windows computers
2. Create dataset
3. Learn collected DLLs with LSTM and create model
Learning phase (Test environment)
Deep learning
Analyst PC
Windows computers
4. Collect DLL information using Sysmon
5. Classify each process into the four malicious tools name or normal
Detection phase (Production environment)

Fig. 1 Overview of the proposed method.

Table 3 Sections of the proposed method.

	Phase	Overview	Section
1	Learning	Collect DLL information using Sysmon	4.3
2	Learning	Create the dataset	4.4
3-1	Learning	Learn collected DLLs with LSTM and create the model	4.5
3-2	Learning	Learn collected DLLs with SVM and Random Forest and create the model	4.6
4	Detection	Collect DLL information using Sysmon	4.3
5	Detection	Classify each process into each malicious tool or normal process using the model	4.7

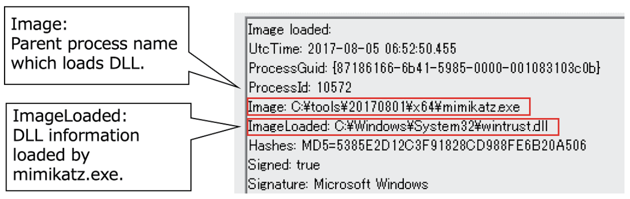


Image:
Parent process name which loads DLL.
ImageLoaded:
DLL information loaded by mimikatz.exe.

```
Image loaded:
UtcTime: 2017-08-05 06:52:50.455
ProcessGuid: {87186166-6b41-5985-0000-001083103c0b}
ProcessId: 10572
Image: C:¥tools¥20170801¥x64¥mimikatz.exe
ImageLoaded: C:¥Windows¥System32¥wintrust.dll
Hashes: MD5=5385E2D12C3F91828CD988FE6B20A506
Signed: true
Signature: Microsoft Windows
```

Fig. 2 An example of Event ID 7.

4.3 Collecting DLL Information Using Sysmon

This section describes how to collect DLL information. The proposed method needs to install Sysmon on the target computers to collect DLL information. In this research, we use Sysmon v6.03. Sysmon is a Windows service that monitors and logs system activity to the Windows Event Log and is useful for recording detailed traces of Windows OS. The Windows OS records activities in the Event Log. However, detailed information on processes including DLL information is not recorded under the default configuration. By using Sysmon, it is possible to record detailed information about the executed processes and loaded DLLs in Event ID 7. **Figure 2** shows an example of Event ID 7. The "Image" field indicates the parent process name. The "ImageLoaded" field indicates the child process including DLLs. In this example, we find out that Mimikatz.exe loads wintrust.dll.

4.4 Creating the Dataset

This section describes how to create the dataset from the DLL information that was collected. The dataset consists of the names

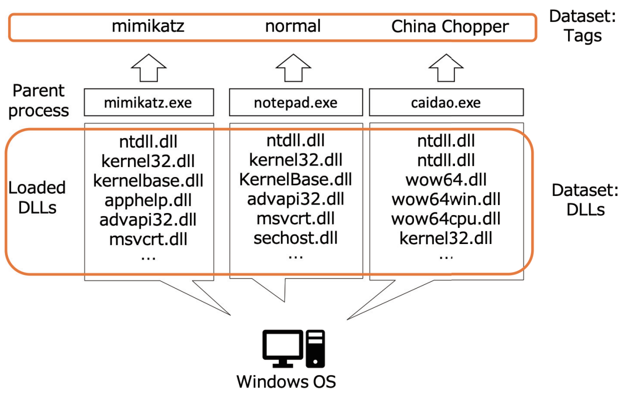

mimikatz
normal
China Chopper
Dataset: Tags
Parent process
mimikatz.exe
notepad.exe
caidao.exe
Loaded DLLs
ntdll.dll kernel32.dll kernelbase.dll apphelp.dll advapi32.dll msvcrt.dll ...
ntdll.dll kernel32.dll KernelBase.dll advapi32.dll msvcrt.dll sechost.dll ...
ntdll.dll ntdll.dll wow64.dll wow64win.dll wow64cpu.dll kernel32.dll ...
Dataset: DLLs
Windows OS

Fig. 3 Method for creating dataset.

Table 4 Windows versions.

Windows version
Windows Server 2016 (x64)
Windows 10 (x64)
Windows Server 2012 (x64)
Windows 8.1 (x64)
Windows Server 2008 R2 (x64)
Windows 7 (x64)

of DLLs loaded by each process extracted from Sysmon logs. The names of DLLs should be kept in the order in which they were loaded. Specifically, the following preprocessing is needed to extract DLLs from event logs and create the dataset.

(1) Export Sysmon logs as text files.
(2) Extract loaded DLLs from the "Image loaded" field in Event ID 7.
(3) Group extracted DLLs by the parent process name ("Image" field).
(4) Concat the name of DLLs loaded by each process in the order of loading.
(5) Extracted DLLs are tagged for supervised learning. DLLs loaded by malicious tools are tagged with each attack tool's process name ("China Chopper," "Mimikatz," "PowerShell Empire" and "HUC Packet Transmitter"). Other DLLs are tagged as "normal." (**Fig. 3**)

In this study, our test environment consists of 5 operators and computers shown in **Table 4** which exemplifies a small organization. We conduct the following operations to collect a training dataset for five days.

- Conduct typical office work is shown in **Table 5**. The same number of "operation order" means that the corresponding operations were operated in random order.
- Run malicious tools shown in **Table 6**. We use different versions of malicious tools for the test data and training data. Several old versions of tools are used for the training data and newer versions are used for test data.

Table 7 shows an example of the dataset. The number of dataset(process) is shown in **Table 8**. The number of unique DLLs is 2324 in the training dataset.

4.5 Learning with the DLL Order Consideration

This section describes the learning algorithm of the proposed method. This method analyzes the DLL name and the order in which they were loaded by each process using deep learning. One of the methods of learning the dataset while considering the order of words is Recurrent Neural Network (RNN) and Long short-term memory (LSTM) [25]. They consider the order of the word sequences. It is also used for detecting anomalies or suspicious activities from time-series data such as logs,

Table 5 Operations for creating normal dataset.

Operation order	Executed works	OS
1	Power on	Client and Server OS
2	Log on	Client and Server OS
3	Start and use Outlook (send e-mail)	Client OS
3	Start and use Microsoft Word	Client OS
3	Start and use Microsoft Excel	Client OS
3	Start and use Microsoft PowerPoint	Client OS
3	Access to the file on the file server with explorer	Client and Server OS
3	Browse the Internet with Internet Explorer and Google Chrome (Google Chrome is used only for evaluation)	Client OS
3	Start and use the command prompt	Client and Server OS
3	Start and use the PowerShell	Client and Server OS
3	Windows Update	Client and Server OS
3	Connected from another computer with remote desktop service	Client and Server OS
4	Log off	Client and Server OS
5	Power off	Client and Server OS

Table 6 Dataset of malicious tools for training.

Tool name	Version
China Chopper	20141213 20111116
Mimikatz	2.1.1 (Aug. 1, 2017) 2.1 (May 1, 2016) 2.0 alpha (May 2, 2015)
PowerShell Empire	2.0 2.3
HUC Packet Transmitter	2003-10-20

Table 7 Example of the dataset.

DLLs	Tags
EppManifest.dll EppManifest.dll sendmail.dll sendmail.dll mydocs.dll audiodev.dll ...	normal
ntdll.dll kernel32.dll kernelbase.dll advapi32.dll msvcrt.dll sechost.dll rpcrt4.dll user32.dll ...	normal
ntdll.dll kernel32.dll kernelbase.dll apphelp.dll advapi32.dll msvcrt.dll sechost.dll ...	mimikatz
ntdll.dll kernel32.dll kernelbase.dll advapi32.dll msvcrt.dll sechost.dll rpcrt4.dll user32.dll ...	normal
ntdll.dll ntdll.dll wow64.dll wow64win.dll wow64cpu.dll kernel32.dll kernel32.dll ...	China Chopper

Table 8 The number of process for training.

	The number of process
China Chopper	14
Mimikatz	27
PowerShell Empire	12
HUC Packet Transmitter	20
Normal data	5246

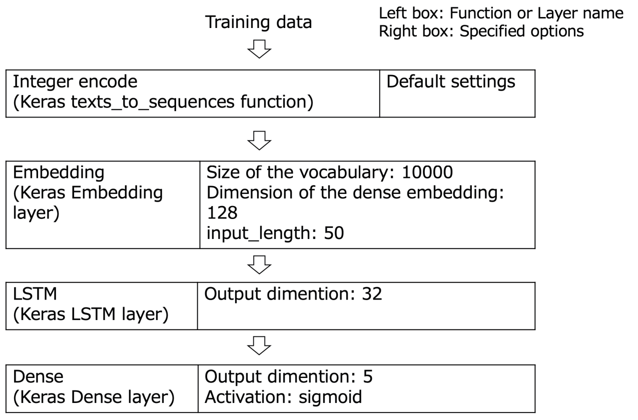


Fig. 4 LSTM flow diagram.

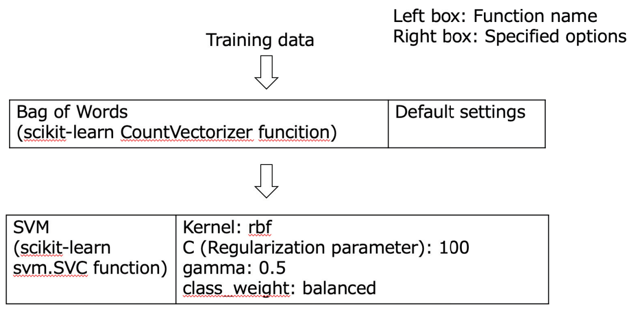

Training data
Left box: Function name
Right box: Specified options
Bag of Words
(scikit-learn CountVectorizer funcition)
Default settings
SVM
(scikit-learn svm.SVC function)
Kernel: rbf
C (Regularization parameter): 100
gamma: 0.5
class_weight: balanced

Fig. 5 BoW with SVM flow diagram.

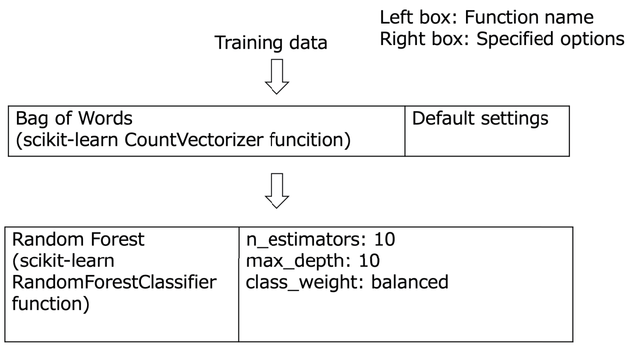


Fig. 6 BoW with RF flow diagram.

Table 9 Dataset of Malicious Tools for Test.

Tool name	Version(test data)
China Chopper	20160622
Mimikatz	2.2.0 (July 20, 2019)
PowerShell Empire	2.5
HUC Packet Transmitter	2003-10-20

Table 10 The number of process for test.

Tool name	The number of process
China Chopper	14
Mimikatz	11
PowerShell Empire	8
HUC Packet Transmitter	18
Normal data	3907

packets etc. [26], [27], [28], [29]. In many cases, LSTM achieves higher accuracy than RNN. Ref. [28] compares them, and LSTM showed better results than RNN. Therefore Our method learns the DLLs and order of the DLL loading using LSTM. LSTM is used for analyzing sequences of data, such as syntactic analysis, and it solves the vanishing gradient problem and can analyze long input data. The LSTM flow diagram and hyperparameters used in the proposed method are shown in **Fig. 4**. We evaluate hyperparameters "Output dimension in LSTM layer" and "Epoch" using grid search. Described hyperparameters in Fig. 4 are the best value obtained from the evaluation result in Section 5.2.2. The training dataset mentioned in Section 4.4 is the time-series DLLs loaded by each process. The proposed method encodes them into an integer sequence for LSTM. Encoded data is processed by the Embedding layer to vectorize data and is connected to the LSTM layer. Finally, the data is classified into five dimensions by the Dense layer.

4.6 Learning without the DLL Order Consideration

To evaluate the DLL load order's contribution, we also conduct a similar examination using the Bag of Words method. Bag of Words keeps multiplicity but does not consider the order of words. It is also used for analyzing logs [30]. One of the methods of learning the dataset without considering the order of words is the combination of the simple Bag of Words and machine learning such as SVM or Random Forest. Bag of Words analyzes frequency of occurrence for each word and extracts word feature. In this research, we classify the extracted feature using SVM (after this, referred to as BoW with SVM) and Random Forest (after this, referred to as BoW with RF). Regarding BoW with SVM, the diagram is shown in **Fig. 5**. We specify "class weight" hyperparameter to "balanced" because of the imbalanced dataset. We also evaluate hyperparameters "C" and "gamma" using grid search. Regarding BoW with RF, the diagram is shown in **Fig. 6**. We set "class weight" hyperparameter to "balanced" and evaluate "n_estimators" and "max_depth" using grid search.

4.7 Detection

In the detection phase, our method collects DLLs on the production environment in the same way with Section 4.3 the learning phase. Then collected DLLs per process is analyzed with the learning model created in Section 4.5 or Section 4.6 and each process is classified as four malicious tools (China Chopper, Mimikatz, PowerShell Empire, HUC Packet Transmitter) or a normal process.

5. Evaluation

This chapter describes the evaluation of the effectiveness of the proposed method.

5.1 Evaluation of Detection Rate

This section describes the evaluation metric of the proposed method. We evaluate whether the proposed method can detect startup of the malicious tools correctly.

The test data of malicious tools is shown in **Table 9**. The number of test data is shown in **Table 10**. For creating a normal test dataset, operators conduct a typical operation as shown in Table 5 for five days. For creating the dataset of malicious tools, we run newer version malicious tools shown in Table 9 which has not been trained. Regarding HUC Packet Transmitter, only one version is released. Thus we run the same version of HUC Packet

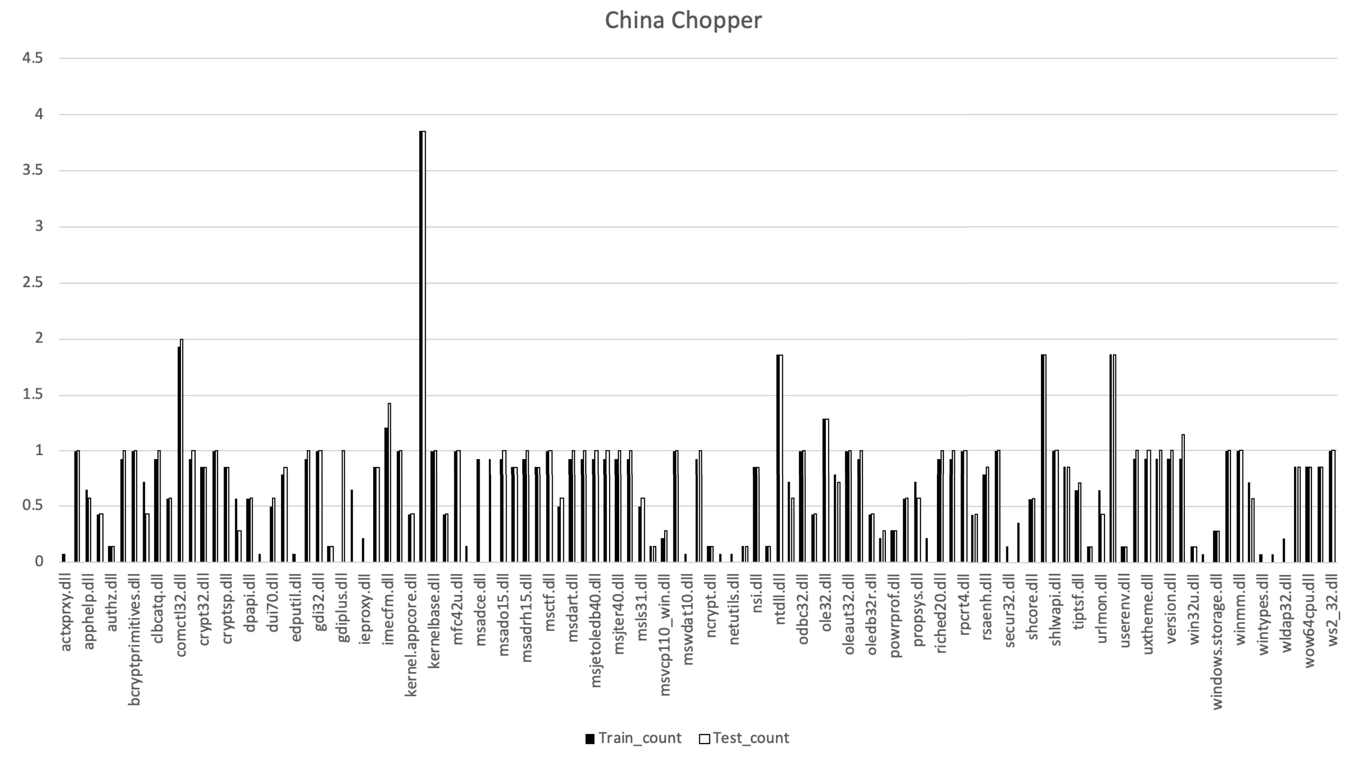


Fig. 7 DLL distribution of China Chopper.

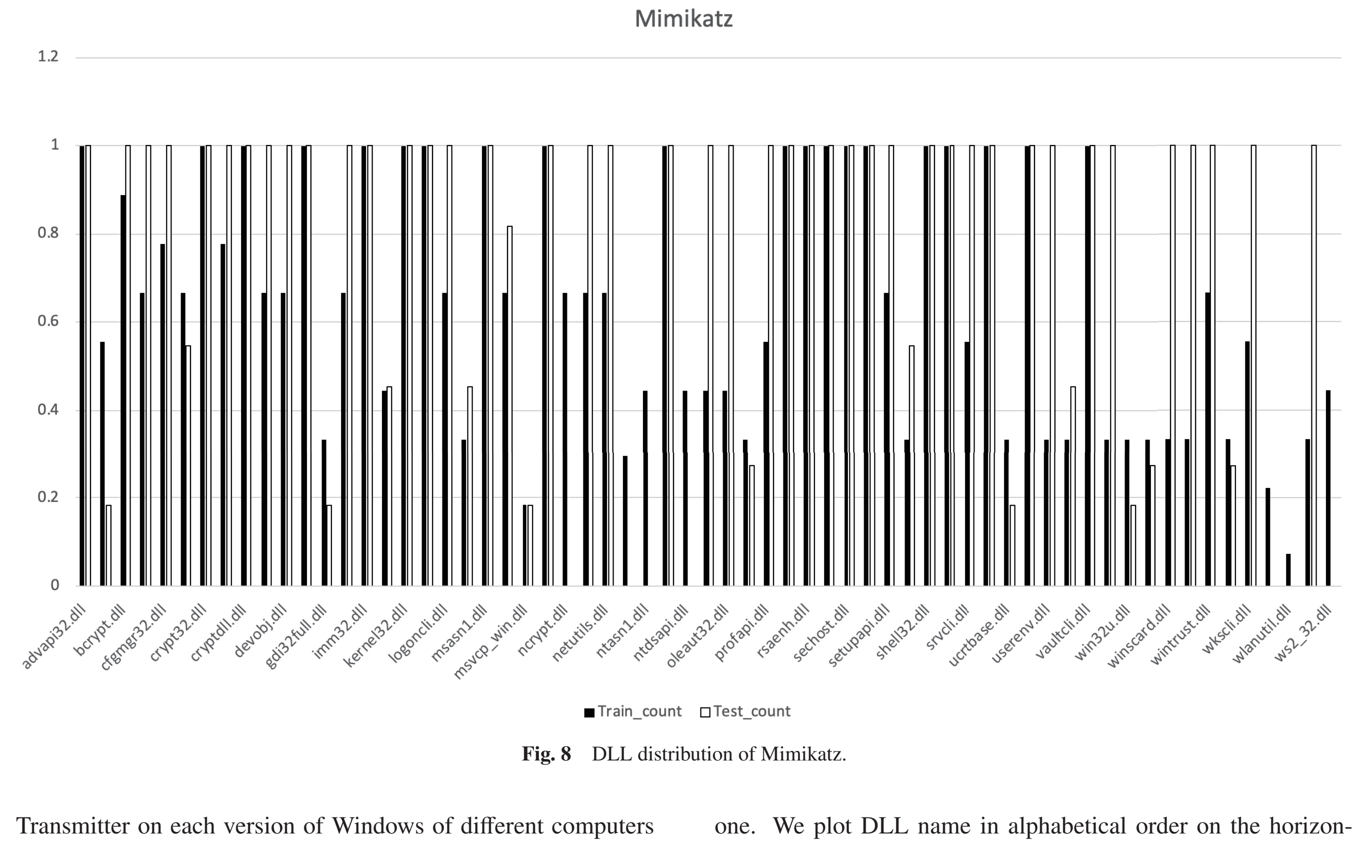


Fig. 8 DLL distribution of Mimikatz.

Transmitter on each version of Windows of different computers from computers used in the training phase. The number of unique DLLs is 1773 in the test dataset.

We investigated the distribution of malicious tools' DLLs of training dataset and test dataset. The number of loaded DLLs is divided by the total process number to take an average of the frequency of occurrence. Note that some process loads the same DLL more than two. Therefore some DLL's average exceeds one. We plot DLL name in alphabetical order on the horizontal axis and average of the frequency of occurrence on the vertical axis. The distribution of malicious tools' DLLs is shown in each **Figs. 7**, **8**, **9**, and **10**. The root mean square of the difference in the frequency of occurrence between training and test data is shown in **Table 11**. As a result, we found that the distribution of malicious tools' DLLs is different between training datasets and test datasets because of the difference in the environment such as

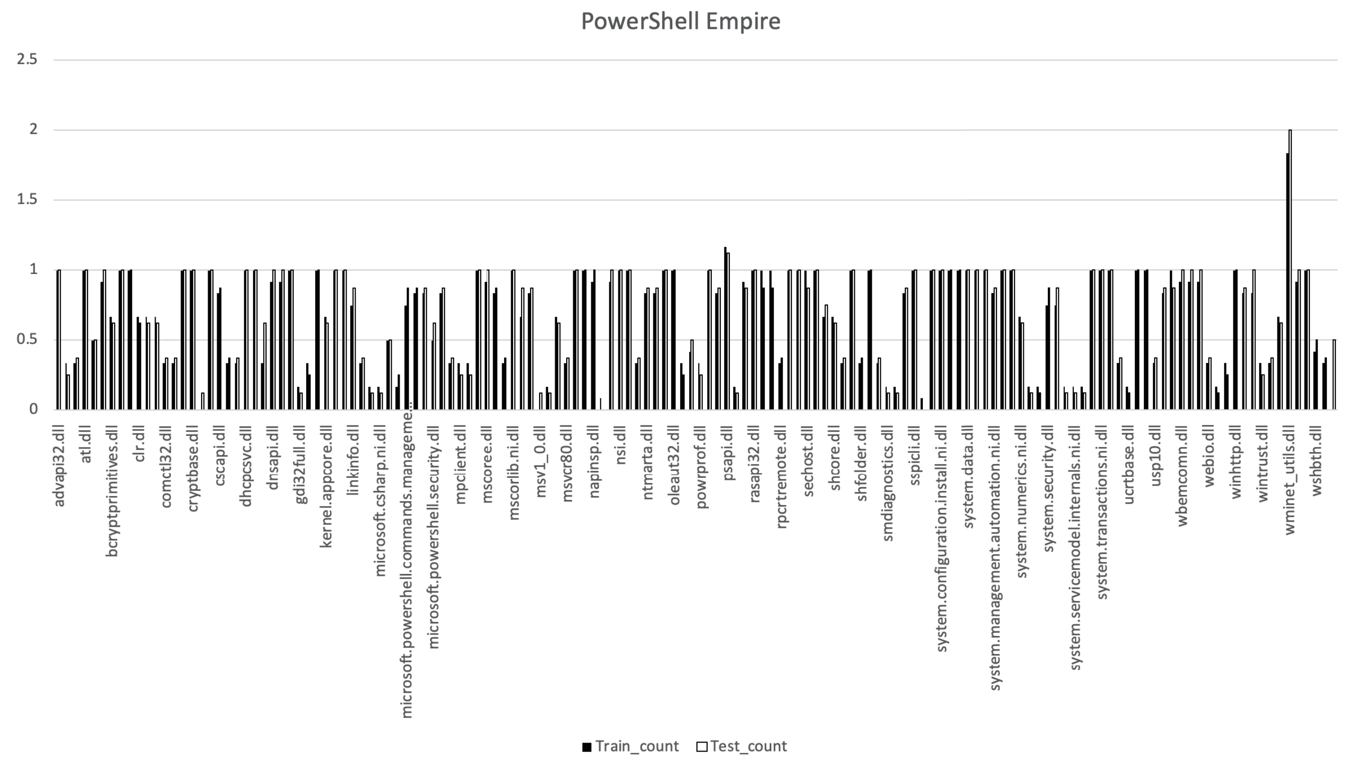


Fig. 9 DLL distribution of PowerShell Empire.

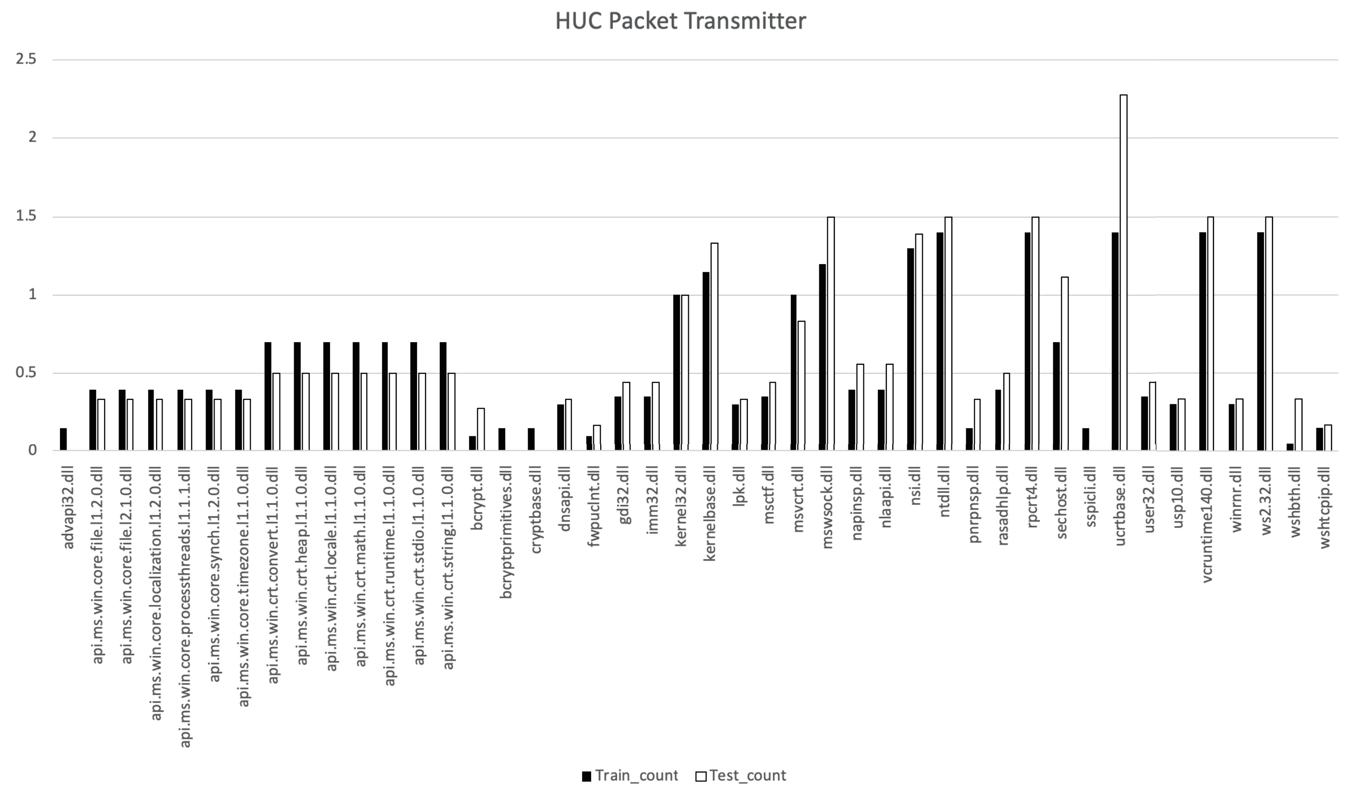


Fig. 10 DLL distribution of HUC Packet Transmitter.

tools' version.

The decision criteria for detection metric is as follows.

- True Positive (TP): The number of processes which the proposed method correctly predicts as each attack tool.
- False Positive (FP): The number of processes which the proposed method incorrectly predicts as attack tools.
- True Negative (TN): The number of processes which the proposed method correctly predicts as normal.
- False Positive (FN): The number of processes which the proposed method incorrectly predicts as normal.
- Recall = TP/(TP + FN)
- Precision = TP/(TP + FP)

Table 11 Root mean square of the difference between training and test data.

Tool name	Root mean square
China Chopper	0.19
Mimikatz	0.31
PowerShell Empire	0.08
HUC Packet Transmitter	0.20

Table 12 Average of detection rate.

LSTM dimension	Recall	Precision	F value
16	95.00%	98.67%	96.80%
32	97.45%	97.29%	97.37%
64	94.67%	93.47%	94.07%

- F value = (2 ∗ Recall ∗ Precision)/(Recall + Precision)

5.1.1 Evaluation Method with DLL Order Consideration

We evaluate the recall, precision, and F value using the LSTM model mentioned in Section 4.5. We evaluate the following variation of hyperparameter.

- Output dimension in LSTM layer: 16, 32, 64
- Epoch: 20, 40, 60, 80, 100

5.1.2 Evaluation Method without DLL Order Consideration

We evaluate the method BoW with SVM and BoW with RF explained in Section 4.6 to investigate the contribution of the order of DLL loading. We evaluate the following variation of hyperparameter for BoW with SVM as following.

- C: 0.1, 1, 10, 100
- gamma: 0.01, 0.1, 0.5, 1.0, 5.0

We evaluate the following variation of hyperparameter for BoW with RF as following.

- n_estimators: 1, 10, 20
- max_depth: 1, 5, 10, 20

5.1.3 Evaluation Result with DLL Order Consideration

We evaluated the detection rate with the proposed method mentioned in Section 5.1.1 using test data. **Table 12** shows the average of recall, precision, and F value per every output dimension in the LSTM layer with 60 epochs. We took ten trials average since the detection rate fluctuated about 5%. As a result, when the output dimension was 32, our proposed method achieved the highest detection rate.

From the above result, we evaluated the variation of epoch with output dimension 32. **Figure 11** shows the ten trials average of recall, precision, and F value per every epoch 20, 40, 60, 80, 100 using the same dataset. As a result, when the epoch was 60, our proposed method achieved the highest detection rate. From above, 32 output dimensions and 60 epochs achieved the following highest detection rate.

- Recall: 97.45%
- Precision: 97.29%
- F value: 97.37%

The method achieved 100% recall, precision, and F value rate once in 25 times. Although, this perfect result could fit only the test data.

The detection rate decreased by more than 60 epochs and it seems overfitting. One of the methods to avoid overfitting is to use a wider variety of datasets [31].

5.1.4 Evaluation Result without DLL Order Consideration

We evaluated the detection rate with the proposed method men-

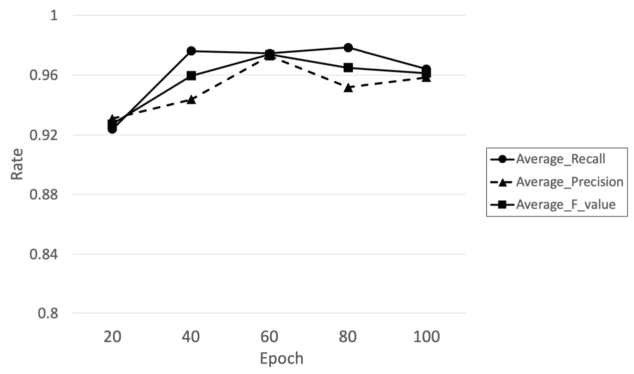


Fig. 11 The detection rate of the proposed method.

tioned in Section 5.1.2 using test data. We took an average of the detection rate of ten trials since the detection rate fluctuated about 5%.

Regarding The BoW with SVM, when the hyperparameter C = 100 and gamma = 0.5, the proposed method achieved the following highest detection rate.

- Recall: 45.60%
- Precision: 100.00%
- F value: 62.64%

Regarding The BoW with RF, when the hyperparameter n_estimator = 10 and max_depth = 10, the proposed method achieved the following highest detection rate.

- Recall: 94.60%
- Precision: 100.00%
- F value: 97.23%

As a result, we found out that the detection rate of the BoW with SVM was lower than LSTM. However, BoW with RF was comparable to LSTM.

5.2 Evaluation of Scalability

This section describes the evaluation of the scalability of the proposed method. DLLs loaded by parent processes, including malicious tools, can be changed in the future. Therefore we evaluate whether the proposed method can detect malicious tools correctly even if loaded DLLs are changed.

5.2.1 Evaluation Method

We insert or/and remove DLLs randomly to the test data assuming that loaded DLLs are changed in the future. First, we investigate the rate of change for DLLs that depend on the version of the attack tools. The concept for the rate of change of DLLs is shown in **Fig. 12**.

- China Chopper: We investigated the difference in loaded DLLs in China Chopper 20111116, 20141213, and 20160622. As a result, the loaded DLLs were almost 100% insert and remove between 20111116 and 20141213. However, in 20160622, almost all DLLs seem to revert to 20111116. The rate of change (insert and remove) between 20111116 and 20160622 is only 1.23%.
- mimikatz: We investigated the difference in loaded DLLs in mimikatz 2.1.1 (Aug. 1, 2017), 2.1 (May 1, 2016), 2.0 alpha (May 2, 2015), and 2.2.0 (July 20, 2019). As a result, about 10% insert or remove were found in a year on average.
- PowerShell Empire: We investigated the difference in loaded

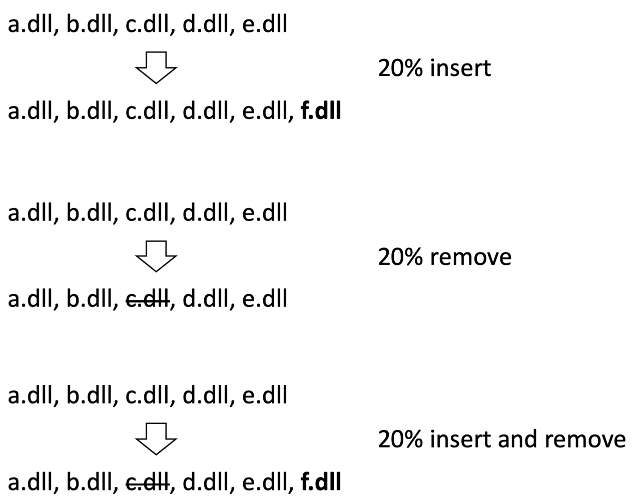

a.dll, b.dll, c.dll, d.dll, e.dll
20% insert
a.dll, b.dll, c.dll, d.dll, e.dll, f.dll
a.dll, b.dll, c.dll, d.dll, e.dll
20% remove
a.dll, b.dll, ~~c.dll~~, d.dll, e.dll
a.dll, b.dll, c.dll, d.dll, e.dll
20% insert and remove
a.dll, b.dll, ~~c.dll~~, d.dll, e.dll, f.dll

Fig. 12 Rate of change of DLL loading.

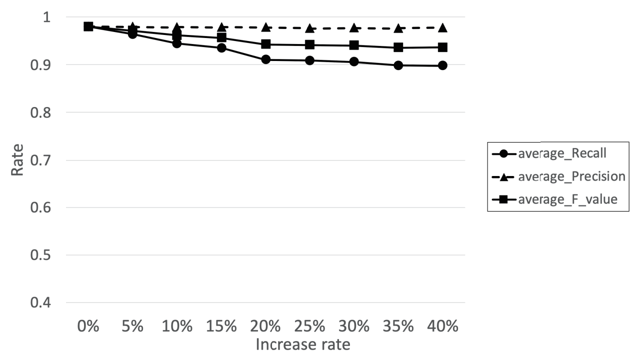


Fig. 13 The detection rate of the LSTM (inserting DLLs).

DLLs in PowerShell Empire 2.0, 2.3, and 2.5. As a result, about 10% to 15% of DLLs were inserted or removed.

Based on the results above, we examine the following conditions.

(1) Insert DLLs at random positions in the test data. The DLLs to insert are randomly selected from the entire loaded DLLs list.

(2) Remove random DLLs at random positions from test data.

(3) Conduct the above both 1, 2 simultaneously.

We evaluate the detection rate of the proposed method when the percentage of changed DLLs is varied from 5% to 40% in 5% increments. For evaluation, we use a model, whose detection rate was Recall 98.0%, Precision 98.0%, F value 98.0%: the closest to the average detection rate shown in Table 12. We evaluate the recall and precision rate by changing DLLs between rates from 5% to 40%, then calculate the average in 100 times of measurement. The evaluation environment is Windows 10 (x64).

5.2.2 Evaluation Result with DLL Order Consideration

We evaluated the scalability of the proposed method mentioned in Section 4.5. **Figure 13** shows the average of the recall and precision rate when DLLs were inserted. According to the result, the recall rate decreased as the percentage of the inserted DLLs increased. Although the proposed method kept more than a 90% recall rate if the percentage of the inserted DLLs was 40% or less.

Figure 14 shows the average of the recall and precision rate when DLLs were removed. According to the result, the recall rate decreased as the percentage of the removed DLLs increased.

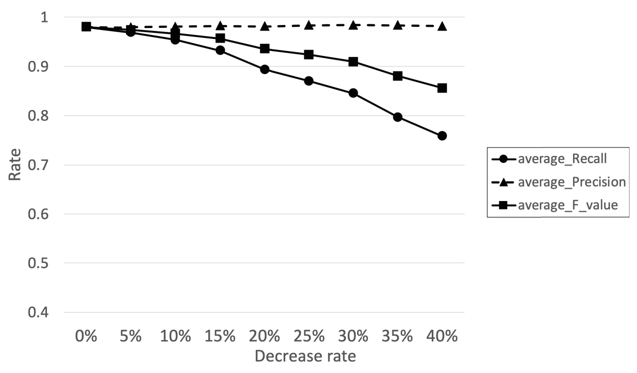


Fig. 14 The detection rate of the LSTM (removing DLLs).

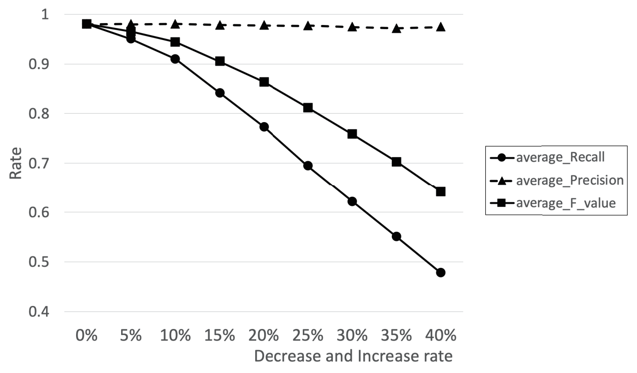


Fig. 15 The detection rate of the LSTM (inserting and removing DLLs).

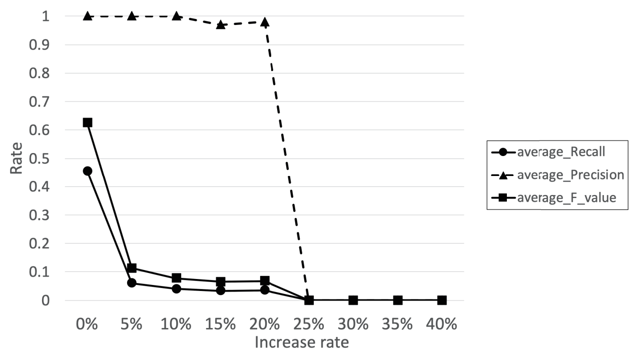


Fig. 16 The detection rate of the BoW with SVM (inserting DLLs).

The proposed method kept more than a 90% recall rate if the percentage of the removed DLLs was 15% or less.

Figure 15 shows the average of the recall and precision rate when DLLs were inserted and removed. According to the result, the recall rate decreased as the percentage of the inserted and removed DLLs increased. The proposed method kept more than a 90% recall rate if the percentage of the inserted and removed DLLs was 10% or less.

5.2.3 Evaluation Result without DLL Order Consideration

We evaluated the scalability of the proposed method mentioned in Section 4.6. Regarding BoW with SVM, **Figs. 16**, **17**, and **18** show the detection rates for each DLL "inserted," "removed," and "inserted and removed." When inserting or removing any DLLs, the number of true positive became zero, therefore Recall, Precision, and F value were calculated as 0%.

Regarding BoW with RF, **Figs. 19**, **20**, and **21** show the detection rate for each DLL "inserted," "removed," and "inserted and removed." When the percentage of inserting/removing DLLs was

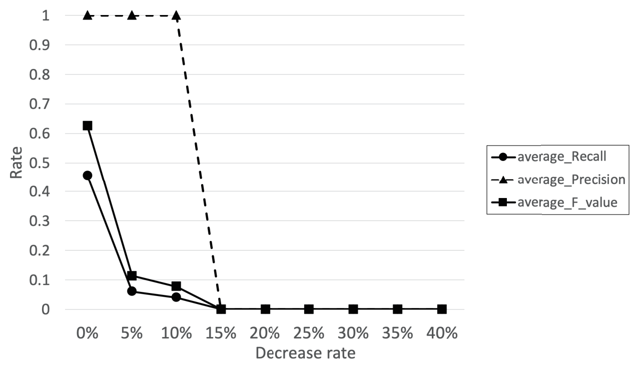


Fig. 17 The detection rate of the BoW with SVM (removing DLLs).

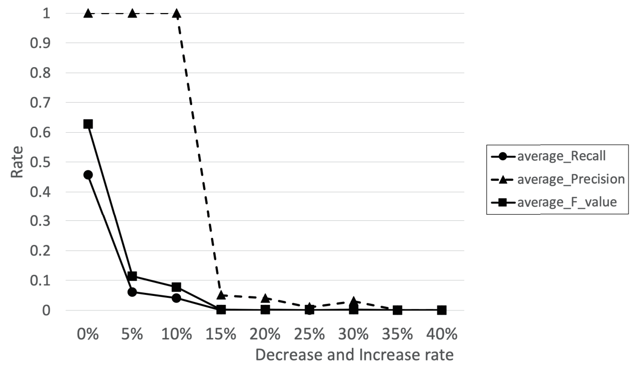


Fig. 18 The detection rate of the BoW with SVM (inserting and removing DLLs).

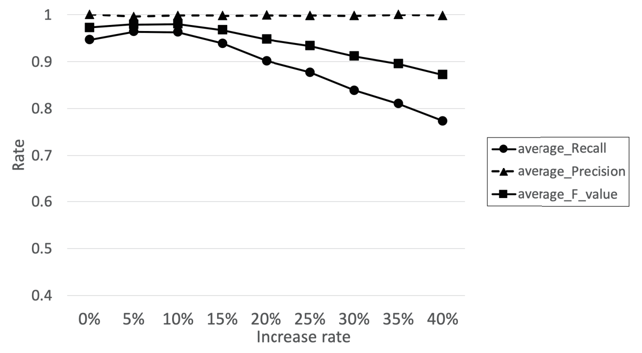


Fig. 19 The detection rate of the BoW with RF (inserting DLLs).

less than 15%, the detection rate was comparable with LSTM. Although, when the percentage of inserting/removing DLLs was 15% or more, the detection rate was worse than LSTM. For instance, when inserting/removing DLLs was 15%, the F value was 90.46% in LSTM and 89.96% in BoW with RF.

From these results, we found out that the proposed method using LSTM has higher robustness than BoW with SVM and BoW with RF. However if DLLs are drastically changed in the future release, we should recreate the dataset and re-evaluate.

5.3 Evaluation of Effect on System Performance

We evaluated the impact on system performance caused by collecting DLLs using the Sysmon. The evaluation environment is Windows server 2008 R2 with a Domain Controller feature (about 30 users access) in our small office environment. The effect of the target computer resources by collecting DLLs using the Sysmon is shown in **Table 13**.

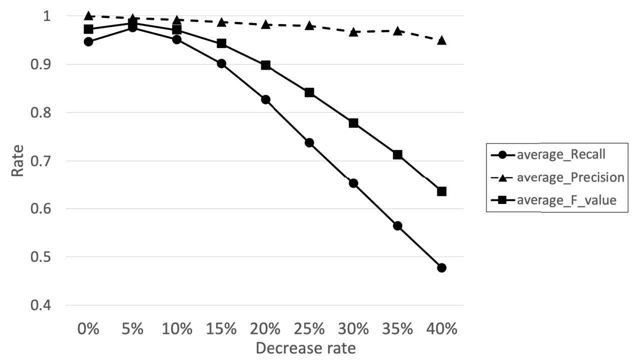


Fig. 20 The detection rate of the BoW with RF (removing DLLs).

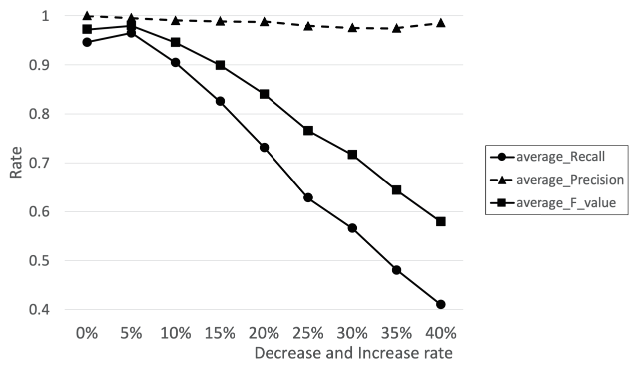


Fig. 21 The detection rate of the BoW with RF (inserting and removing DLLs).

Table 13 Affection on system performance.

Memory usage	increased from 21% to 24% on 8GB Memory
CPU usage	Not changed
Disk usage	increased 600MB per day. 95% of event log count was Sysmon Event ID 7

5.4 Discussion on False Detection

We investigated false detections focusing on the result where both recall and precision achieved 96% shown in Section 5.1.3. The total number of false detections occurred in ten trials per every epoch 20, 40, 60, 80, 100 (50 trials in total) using the same dataset is shown in **Table 14**. For example, the PowerShell Empire caused six false negatives out of 50 trials. We described a breakdown of false detection in **Table 15**. False negatives only occurred in the detection of the PowerShell Empire. In the case of false positives, we described the true process name in parentheses.

False positives occurred in the detection of all four tools, but especially the detection of China Chopper and mimikatz was notable.

The following processes were incorrectly classified as mimikatz:

- shellexperiencehost.exe
- chrome.exe
- svchost.exe

The following processes were incorrectly classified as China Chopper:

- shellexperiencehost.exe
- iexplore.exe
- svchost.exe

For analyzing the above process which caused false positives,

Table 14 The total number of false detections occurred in ten trials per every epoch.

		Prediction				
		China Chopper	PowerShellEmpire	HTC PacketTransmitter	Mimikatz	Normal
True Value	China Chopper	0	0	0	0	0
	PowerShell Empire	0	0	0	0	6
	HTC Packet Transmitter	0	1	0	0	0
	Mimikatz	1	0	0	0	0
	Normal	3	1	1	3	0

Table 15 The breakdown of the false detection.

No.	Epoch	Recall	Precision	F value	True value (True Process)	Prediction
1	40	100%	98.1%	99.0%	Normal (svchost.exe)	PowerShellEmpire
2	40	98.0%	100%	98.8%	PowerShellEmpire	Normal
3	40	98.0%	98.0%	98.0%	Normal (shellexperiencehost.exe)	HTC PacketTransmitter
					PowerShellEmpire	Normal
4	60	98.0%	98.0%	98.0%	Normal (shellexperiencehost.exe)	China Chopper
					PowerShellEmpire	Normal
5	60	100%	100%	100%	-	-
6	60	98.0%	100%	99.0%	PowerShellEmpire	Normal
7	60	98.0%	98.0%	98.0%	PowerShellEmpire	Normal
					Normal (shellexperiencehost.exe)	Mimikatz
8	80	98.0%	98.0%	98.0%	Normal (chrome.exe)	Mimikatz
					PowerShellEmpire	Normal
9	100	100%	96.2%	98.1%	Normal (iexplore.exe)	China Chopper
					Mimikatz	China Chopper
10	100	100%	96.2%	98.1%	HTC PacketTransmitter	PowerShellEmpire
					Normal (svchost.exe)	Mimikatz
11	100	100%	100%	100%	-	-
12	100	100%	98.1%	99.0%	Normal (svchost.exe)	China Chopper

Table 16 The processes that caused false positives.

Process name	The total number of the process	The number of false positive
svchost.exe	217	3
chrome.exe	374	1
iexplore.exe	83	1
shellexperiencehost.exe	11	3

we described the total number of each process which caused false positives in the test dataset and the number of its false positive in **Table 16**. As a result, regarding svchost.exe, chrome.exe, and iexplore.exe, the ratio of false-positives is less than 2%. As a result, only a few percents of increase of memory usage occurred. It can be considered that the result could be without any practical problems.

False positives were not 100% reproducible since DLLs loaded by each process were different depending on the situation. For instance, we ran the same process (Internet Explorer) on the same machine. However, the loaded DLLs were slightly different from each time.

It is assumed that the frequency of occurrence of each DLL and order of DLLs is related to the false detections, but we cannot find concrete reasons as of yet. We will investigate it in future works.

6. Conclusion

In targeted attacks, attackers tend to avoid detection by changing file names or rebuilding tools. Therefore, detecting malicious tools is difficult. In this research, we propose a detection method of these malicious tools by analyzing DLL information using deep learning, taking the order in which the DLLs were loaded into consideration. The proposed method could detect four tools: Mimikatz, PowerShell Empire, and HUC Packet Transmitter with high accuracy even if the file name is changed or tools are rebuilt. Furthermore, the proposed method can detect malicious tools with more than a 90% detection rate, even if about 10% of loaded DLLs are changed in the future release. In this research, we evaluated four malicious tools, but our approach could be extended to other malicious tools. BoW with SVM and BoW with RF was able to classify the malicious tools. Therefore, it can be concluded that the DLL name itself has unique characteristics to classify malicious tools. However, the proposed method using LSTM has a higher robustness than BoW with SVM and BoW with RF. Hence, we clarified that not only DLL itself but also the order of loading DLLs are the unique characteristics to classify the malicious tools.

For future work, we will analyze false positives, and furthermore expand the proposed method to detect other malicious activities in targeted attacks.

Finally, we have published an implementation of this proposed method in our Github [32].

References

[1] CISA: Publicly available tools seen in cyber incidents worldwide (2018).

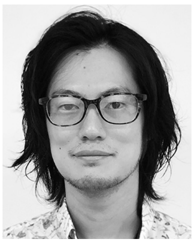
[2] Sjarif, N.N.A., Chuprat, S., Mahrin, M.N., Ahmad, N.A., Ariffin, A., Senan, F.M., Zamani, N.A. and Saupi, A.: Endpoint detection and response: Why use machine learning? *2019 International Conference on Information and Communication Technology Convergence* (*ICTC*), pp.283–288 (Oct. 2019).

[3] Ghafir, I. and Prenosil, V.: Malicious file hash detection and drive-by download attacks, *Proc. Second International Conference on Computer and Communication Technologies*, pp.661–669, Springer (2016).

[4] Mulder, J.: The sans institute: Mimikatz overview, defenses and detection (2016), available from ⟨https://www.sans.org/reading-room/whitepapers/detection/mimikatz-overview-defenses-detection-36780⟩.

[5] RenditionSec: Antivirus isn't dead, but you need monitoring too (2017), available from ⟨https://blog.renditioninfosec.com/2017/11/antivirus-isnt-dead-but-you-need-monitoring-too/⟩.

[6] Ussath, M., Jaeger, D., Cheng, F. and Meinel, C.: Advanced persistent threats: Behind the scenes, *2016 Annual Conference on Information Science and Systems* (*CISS*), pp.181–186 (2016).
[7] Juwono, J.T., Lim, C. and Erwin, A.: A comparative study of behavior analysis sandboxes in malware detection (2015).
[8] Kesavan, K., Bannakkotuwa, S., Wickramanayake, V.V.Y., De Silva, M.P.D.H., Fernando, J.M.D., Sampath, K. and Rupasinghe, P.: Clustermal: Automated malware analysis with clustering, anomaly detection and classification of existing and new behavioral analysis (2016).
[9] Willems, C., Holz, T. and Freiling, F.: Toward automated dynamic malware analysis using cwsandbox, *IEEE Security Privacy*, Vol.5, No.2, pp.2–39 (Mar. 2007).
[10] Matsuda, W., Fujimoto, M. and Mitsunaga, T.: Real-time detection system against malicious tools by monitoring dll on client computers, *2019 IEEE Conference on Application, Information and Network Security* (*AINS*), pp.36–41 (Nov. 2019).
[11] Virus total (2018), available from ⟨https://www.virustotal.com⟩.
[12] Active Directory Security: Unofficial guide to mimikatz & command reference (2018), available from ⟨https://adsecurity.org/?page_id=1821⟩.
[13] Narouei, M., Ahmadi, M., Giacinto, G., Takabi, H. and Sami, A.: Dllminer: Structural mining for malware detection, *Sec. Commun. Netw.*, Vol.8, No.18, pp.3311–3322 (Dec. 2015).
[14] Ki, Y., Kim, E. and Kim, H.K.: A novel approach to detect malware based on api call sequence analysis, *International Journal of Distributed Sensor Networks*, Vol.11, No.6, p.659101 (2015).
[15] Gonzalez, L.E. and Vazquez, R.A.: Malware classification using euclidean distance and artificial neural, *2013 12th Mexican International Conference on Artificial Intelligence*, pp.103–108, IEEE (2013).
[16] Ijaz, M., Durad, M.H. and Ismail, M.: Static and dynamic malware analysis using machine learning, *2019 16th International Bhurban Conference on Applied Sciences and Technology* (*IBCAST*), pp.687–691, IEEE (2019).
[17] Macías, M., Barría, C., Acuna, A. and Cubillos, C.: Sgsi support throught malware's classification using a pattern analysis, *2016 IEEE International Conference on Automatica* (*ICA-ACCA*), pp.1–4 (2016).
[18] Duan, Y., Fu, X., Luo, B., Wang, Z., Shi, J. and Du, X.: Detective: Automatically identify and analyze malware processes in forensic scenarios via dlls, *2015 IEEE International Conference on Communications* (*ICC*), pp.5691–5696, IEEE (2015).
[19] Sun, H.-M., Lin, Y.-H. and Wu, M.-F.: Api monitoring system for defeating worms and exploits in ms-windows system, *Information Security and Privacy*, pp.159–170, Springer Berlin Heidelberg (2006).
[20] Liu, S.-T., Huang, H.-C. and Chen, Y.-M.: A system call analysis method with mapreduce for malware detection, *2011 IEEE 17th International Conference on Parallel and Distributed Systems*, pp.631–637, IEEE (2011).
[21] Gong, M., Girkar, U. and Xie, B.: Classifying windows malware with static analysis (2016).
[22] sisoc tokyo, attacktooldetection_sysmon (2019), available from ⟨https://github.com/sisoc-tokyo/attackToolDetection_Sysmon⟩.
[23] Joven, R. and Paz, R.D.: Jbifrost: Yet another incarnation of the adwind RAT (2016).
[24] Mavroeidis, V. and Jøsang, A.: Data-driven threat hunting using sysmon, *Proc. 2nd International Conference on Cryptography, Security and Privacy*, pp.82–88, Association for Computing Machinery (2018).
[25] Hochreiter, S. and Schmidhuber, J.: Long short-term memory, *Neural Computation*, Vol.9, No.8, pp.1735–1780 (1997).
[26] Ando, Y., Gomi, H. and Tanaka, H.: Detecting fraudulent behavior using recurrent neural networks, *Computer Security Symposium* (2016).
[27] Feng, C., Li, T. and Chana, D.: Multi-level anomaly detection in industrial control systems via package signatures and lstm networks, *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks* (*DSN*), pp.261–272, IEEE (2017).
[28] Malhotra, P., Vig, L., Shroff, G. and Agarwal, P.: Long short term memory networks for anomaly detection in time series, *Proc. ESANN 2015*, p.89, Presses universitaires de Louvain (2015).
[29] Fujimoto, M., Matsuda, W. and Mitsunaga, T.: Deep learning wo mochiita struts 2 wo akuyousuru kogeki no bogyo (defending attacks leveraging struts 2 using deep learning), *Digital Practice*, Vol.10, No.2, pp.381–402 (2019).
[30] Salem, M.B. and Stolfo, S.J.: Detecting masqueraders: A comparison of one-class bag-of-words user behavior modeling techniques, *JoWUA*, Vol.1, No.1, pp.3–13 (2010).
[31] Wang, H., Yin, J., Pei, J., Yu, P.S. and Yu, J.X.: Suppressing model overfitting in mining concept-drifting data streams, *Proc. 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp.736–741, Association for Computing Machinery (2006).
[32] sisoc tokyo, attacktooldetection_deeplearning (2019), available from ⟨https://github.com/sisoc-tokyo/attackToolDetection_DeepLearning⟩.



Wataru Matsuda joined NTT WEST, Ltd. in 2006. In 2015, he joined Watch and Warning Group of JPCERT/CC, where he has engaged in information gathering and early warning activities. Now as a Project Researcher of Secure Information Society Research Group, the University of Tokyo, he engages in research on cyber security, especially log analysis for detecting targeted attacks.

Mariko Fujimoto joined NEC Solution Innovators, Ltd. in 2004 and worked for development of software and systems for internal control. In 2015, she joined Watch and Warning Group of JPCERT/CC, where she was engaged in information gathering and early warning activities. Now as a Project Researcher of Secure Information Society Research Group, the University of Tokyo, she is engaged in research on cyber security especially log analysis for detecting targeted attacks.

Takuho Mitsunaga is a Associate Professor at Toyo University. He is also a Research Fellow at Information-technology Promotion Agency in Japan. After completing his degree at Graduate School of Informatics, Kyoto University, He worked at the front line of incident handling and penetration testing at a security vendor. In FY 2010, he led an R&D project of the Ministry of Trade, Economy and Industry (METI) for encryption data sharing system for cloud with an efficient key managing function. He has been a member of Watch and Warning Group of JPCERT/CC since April 2011, where he is engages in cyber attack analysis including APT cases. He has also contributed in some cyber security related books as coauthor or editorial supervisor including "Information Security White Paper 2013."