IEICE TRANS. INF. & SYST., VOL.E101–D, NO.11 NOVEMBER 2018

PAPER Special Section on Information and Communication System Security

Automatically Generating Malware Analysis Reports Using Sandbox Logs

Bo SUN[†,††a)], ***Member***, **Akinori FUJINO**[††b)], ***Nonmember***, **Tatsuya MORI**[††,†††c)], ***Member***, **Tao BAN**[†d)], ***Nonmember***, **Takeshi TAKAHASHI**[†e)], ***and*** **Daisuke INOUE**[†f)], ***Members***

SUMMARY Analyzing a malware sample requires much more time and cost than creating it. To understand the behavior of a given malware sample, security analysts often make use of API call logs collected by the dynamic malware analysis tools such as a sandbox. As the amount of the log generated for a malware sample could become tremendously large, inspecting the log requires a time-consuming effort. Meanwhile, antivirus vendors usually publish malware analysis reports (vendor reports) on their websites. These malware analysis reports are the results of careful analysis done by security experts. The problem is that even though there are such analyzed examples for malware samples, associating the vendor reports with the sandbox logs is difficult. This makes security analysts not able to retrieve useful information described in vendor reports. To address this issue, we developed a system called *AMAR-Generator* that aims to automate the generation of malware analysis reports based on sandbox logs by making use of existing vendor reports. Aiming at a convenient assistant tool for security analysts, our system employs techniques including template matching, API behavior mapping, and malicious behavior database to produce concise human-readable reports that describe the malicious behaviors of malware programs. Through the performance evaluation, we first demonstrate that AMAR-Generator can generate human-readable reports that can be used by a security analyst as the first step of the malware analysis. We also demonstrate that AMAR-Generator can identify the malicious behaviors that are conducted by malware from the sandbox logs; the detection rates are up to 96.74%, 100%, and 74.87% on the sandbox logs collected in 2013, 2014, and 2015, respectively. We also present that it can detect malicious behaviors from unknown types of sandbox logs.

key words: *sandbox logs, malware analysis, automated report generating, natural language processing*

1. Introduction

Computer malware remains a significant threat to our daily lives. According to a report by AV-TEST [1], approximately 390,000 types of new malware are detected daily, and the total number of malware instances detected in 2015 was approximately 470 million. To mitigate such threats, malware analysis is a crucial approach to understand various malware features that can be used to develop and improve malware detection systems. Generally, malware analysis can be categorized into static or dynamic approaches. The dynamic analysis can leverage the actual controlled environment to detect malicious behavior hidden by obfuscation code, whereas the static analysis cannot.

For security analysts, the dynamic analysis is indispensable for analyzing malware samples. When they investigate sandbox logs, i.e. the output of the dynamic analysis, for identifying malicious behaviors, it is inefficient to analyze a large number of sandbox logs manually. Meanwhile, antivirus vendors analyze huge volumes of malware programs and make the analysis reports open to public access via the Internet. In general, malware analysis reports provided by vendors (vendor reports) are written in natural languages and do not include details of various API calls or related arguments. Furthermore, such reports are relatively independent of one another in terms of malware types, thus it is difficult to associate sandbox logs with these vendor reports in terms of API calls and corresponding arguments. Moreover, it is often difficult to extract either the same or different characteristics from these reports. This makes security analysts not able to retrieve useful information described in vendor reports effectively or efficiently when they are analyzing sandbox logs, despite that there are a vast number of vendor reports existed in real world and even more are generated constantly.

To bridge this gap, we developed a system called *AMAR-Generator* to automatically generate malware analysis reports that can precisely describe the malicious behaviors found in sandbox logs based on knowledge presented in vendor reports. The interpretation of the information described in these two types of documents are exploited in the following steps. First, we replace abstract expressions, such as "<random number>.exe," in vendor reports by applying the template matching and then store the extracted templates to a malware behavior database. Second, we obtain API calls and the value names from sandbox logs, using API Behavior Map as a searching method. Then, to create a relative correlation, we adopt the malware behavior database to confirm whether the values of API calls extracted from sandbox logs are malicious or not. Finally, based on matching results from malware behavior database, we leverage description extracted from vendor reports to produce concise human-readable reports about malicious behaviors.

Manuscript received November 14, 2017.
Manuscript revised May 11, 2018.
Manuscript publicized August 22, 2018.

[†]The authors are with the National Institute of Information and Communications Technology, Koganei-shi, 184–0015 Japan.
[††]The authors are with the Department of Communication Engineering, Waseda University, Tokyo, 169–8050 Japan.
[†††]The author is with the Center for Advanced Intelligence Project, RIKEN, Wako-shi, 351–0198 Japan.
a) E-mail: sunshine@nsl.cs.waseda.ac.jp
b) E-mail: fujino@nsl.cs.waseda.ac.jp
c) E-mail: mori@nsl.cs.waseda.ac.jp
d) E-mail: bantao@nict.go.jp
e) E-mail: takeshi_takahashi@nict.go.jp
f) E-mail: dai@nict.go.jp
DOI: 10.1587/transinf.2017ICP0011

The chief contributions of our work are summarized as follows:

- We propose a system called *AMAR-Generator* that automatically generate the concise human-readable reports for security analysts based on the knowledge described in the vendor reports.
- Experimental results demonstrate that from the sandbox logs, AMAR-Generator can automatically detect the malicious behaviors that need to be reported to a security analyst. The detection rates are up to 96.74%, 100%, and 74.87% on the logs collected in 2013, 2014, and 2015, respectively.

To the best of our knowledge, this is the first work that attempts to democratize the generation technology of malware analysis reports and evaluate it in a scientific manner. We note that off-the-shelf sandbox products can also generate a human-readable analysis report. However, such products are a blackbox, and we need to work on a white-box approach. Such an approach may enable us to make the results reproducible for the research community, and lead to the further improvement of the technology.

The rest of this paper is organized as follows. Section 2 presents the dataset used by AMAR-Generator. Section 3 describes the methodology of AMAR-Generator. Section 4 presents performance evaluation of AMAR-Generator. Section 5 summarizes related work and compare them with AMAR-Generator. Section 6 discusses the limitation of AMAR-Generator and future research direction. Finally, Sect. 7 draws the conclusion.

2. Dataset

In this section, we describe the sandbox logs that we used and how we gathered vendor reports from the real world.

2.1 Sandbox Log

We used the FFRI datasets from 2013 to 2015 provided by FFRI, Inc. [2]. The FFRI datasets are also part of the MWS datasets [3] collected by different research institutes and industries in Japan. The number of malware samples was 8,644. FFRI dataset content was represented as JSON files that were the output of the cuckoo sandbox [4]. Malware executed in the cuckoo sandbox is in the PE format with the maximum execution time of each malware test set to 90 seconds. The resulting JSON files contained scan results of Virustotal [5], API calls, network traffic, and registry and file information created or accessed by the malware sample. API calls were arranged in descending order of time, with the information associated with each API call described in detail, including function name, parameters, and category.

2.2 Vendor Reports

Many antivirus vendors, such as Microsoft and Symantec, publish malware analysis reports on their website. We built a crawler to collect the HTML (HyperText Markup Language) pages of Microsoft's reports using their common URL pattern (i.e., http://www.microsoft.com/security/portal/threat/encyclopedia/entry.aspx?Name=“*malware type*”). We identified 1,299 malware types in the FFRI datasets. Then, we changed the malware type in the common URL pattern to crawl 1,678 malware reports. Note that although the writing style and structure of these reports differ among antivirus vendors, By only specifying the extracted information's HTML tags, the proposed system was designed to handle different antivirus vendor reports. In this paper, owing to space limitation, we cannot show all types of vendor reports. Microsoft's reports are widely used in malware analysis; thus, we present Microsoft's reports as an example. Microsoft reports consisted of the following four parts: “Summary,” “what to do now,” “Technical information,” and “Symptoms.” Note that we only considered the Technical information because that component presented details about the malicious behaviors.

3. AMAR-Generator System

In this section, we first present the architecture of our AMAR-Generator system, and then detail its four core components: vendor report preprocessing, malicious behavior extraction from vendor reports, malicious behavior extraction from sandbox logs and malware analysis report generation.

3.1 Overview

An overview of AMAR-Generator is shown in Fig. 1. First, we preprocess the vendor reports to remove noise. Second, we extract malicious behavior from the vendor reports to construct the malicious behavior database. Third, we specify API call value names described in sandbox logs using a API behavior Map and then determine if the API call value names match malicious behavior in the database. Fourth, malicious behaviors detected by the proposed system are used to generate a new malware analysis report.

3.2 Vendor Report Preprocessing

As the vendor reports were created by different people, we

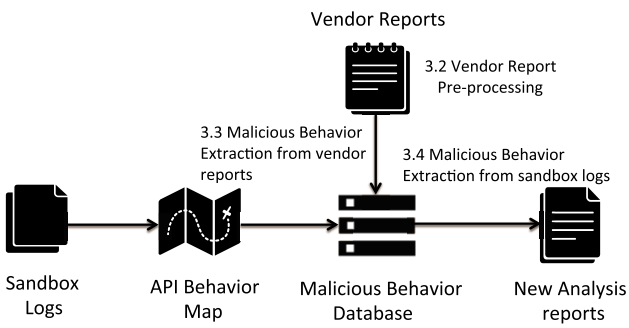

Fig. 1 Overview of the AMAR-Generator system.

Table 1 The list of the procedure of malicious behavior extraction.

NO.	Procedure
1	Transform all text into lowercase.
2	Split the file path using the delimiter "\".
3	If the divided element contains an abstract expression, convert it to a regular expression.
4	Save each element in the database as a nested tag.
5	When saving the final element, create lists of the description of malicious behaviors and malware names as "description" and "malware," respectively, and save the description of malicious behaviors and malware names.
6	If each element has been saved in the database, only add the description of malicious behaviors and the malware names.
7	Split the registry key using the delimiter extbackslash
8	Delete the "https://" and "http://" that exsit in front of domain.
9	Split the character string with the delimiter extbackslashhich is closest to the front of string and save the splited two parts to database.

employed the following preprocessing steps to eliminate differences.

Step1: Extract the Technical information.
Step2: Convert all extracted data to plain text.
Step3: Unify synonym notations.
Step4: Reconstruct the text format.

Step1: Technical Information Extraction. As mentioned in Sect. 2, we focus on the descriptions written in the "Technical Information" part. Note that the "Technical Information" is formatted in HTML and stored between div tags whose id is tab-link-3C, and we use this div tag attribute to extract the "Technical Information".

Step2: Plain Text Transformation. The collected Microsoft analysis reports are formatted as HTML pages. Preprocessing is required because report creators have individual styles relative to tag use. To simplify preprocessing, we converted the HTML pages to plain text by recursively processing the tag types, such as block-level elements and inline elements. Preprocessing proceeded from the head of the child elements in the "Technical Information." In the following, RESULT indicates the text output. HTML elements are composed of three types of child elements, i.e., text, comments, and nested HTML tags.

1. If the element is a comment, delete the comment.
2. If the element is text, concatenate the text to the end of the RESULT.
3. If the element is an HTML tag, Step 2 is performed from the head of the child element.
4. When the element corresponds to block-level elements, i.e., li or br, add a newline character to the end of RESULT.

Step3: Synonym Notation Unification. Due to the different writing style in each report, there are differences in notation even though the same malicious behavior is described. For example, a registry subkey can be written as "In Subkey:" or "To Subkey:." To reduce the time required to extract malicious behavior, we unify the notation by defining a synonym notation list. Part of the synonym notation list is shown in Table 2.

Step4: Text Reconstruction. Descriptive text related to malicious behavior includes target files, directories, processes, destination domains, and the names of related malware. Experts provide additional explanations about malicious behavior at the end of each sentence in such descriptions. Note that such information is not useful for building our malicious behavior database; therefore, we delete the additional explanations prior to extracting malicious behavior using a prefix in additional explanations, such as ", for example", "(e.g," and, "where."

Table 2 A part of synonym notation.

Item	Notation
Registry key	In subkey:, In subkeys:, Under subkey:, Under subkeys:, To subkey:, To subkeys:, Within subkey:, Within subkeys:, The subkey:, The subkeys:
Registry entry name	Sets value:, Sets values:, Modify Registry value:, Adding the value:, Adding registry value:, With value:, With values:, Under value:, Under values:, Changes value:, Modifies value:, Modify value:
Registry entry value	With data:, To data:, From data:
Partition	<drive>, <drive:>, <targeted drive>, <drive root>, <drive letter>

3.3 Malicious Behavior Extraction from Vendor Reports

In this subsection, we present how to extract malicious behavior from vendor reports. Note that Malicious behavior in this paper is defined as a threatening behavior described in Microsoft's malware analysis report. Malicious behavior and IOCs have common elements, such as file paths and registry keys. IOCs are used to indicate an attacker's modus operandi, and recognize behaviors that are exclusively associated with specific malware. On the other hand, threatening behaviors include all activities related to a certain malware. Although some benign behaviors may exist in vendor reports, our system is not designed to detect malicious behaviors from vendor reports. We aim to provide malware information to malware analysts in the form of a report.

3.3.1 File Operation Extraction

If an element of a block is a file path or a file name, we classify it as a file operation. The following rules are used to determine whether the description is a file path or file name.

- A character string starting with an environment variable, as described in the literature [6], [7]

Table 3 The content of Malicious Behavior Database.

Operation	Behavior	Description	Malware Type
File	%windir%\che08.exe	If this worm is executed, this file copies itself to the Windows folder as in the following examples:	worm:win32/koobface.gen!d
Registry	%appdata%\\microsoft \\windows\\ieupdate\\randomname.exe	They can also change the following registry entry so they run each time you start your PC:	trojan:win32/ropest
Network	0.pool.ntp.org	this file connects to the following servers every 20 seconds to send and receive messages:	trojan:win32/necurs.gen!a
Mutex	global\\mp6c3ygukx29gbdk	This threat can create a mutex on your PC. For example:	trojanproxy:bat/dafterdod.a

Table 4 The description method of registry operation.

Description 1	Description 2
In subkey: regkey1	Sets value: value1
...	With data: data1
In subkey: regkeyX	...
Sets value: value1	Sets value: valueX
With data: data1	With data: dataX
...	In subkey: regkey1
Sets value: valueY	...
With data: dataY	In subkey: regkeyY

- A character string starting with either “a~z” and “: \” or “<targeted drive>” and “: \”.
- A character string that ends with “.” as a file extension and does not begin with the registry key prefix.

A character string beginning with either “a~z” and “: \” or “<targeted drive>” and “: \” indicates an external storage device, e.g., the C drive and a USB drive in Windows. We use such strings to determine a file path. We found 19,018 entries beginning with a~z in the file extension. We create a file extension list by referring to the literature [8]. According to the above rule, we extract malicious behavior of file operations from vendor reports. Because four types of operation extraction have common procedures, we summarize all the extraction procedures in Table 1. Regarding file operation, the extraction procedure is from 1 to 6.

3.3.2 Registry Operation Extraction

The following rules are used to determine whether each element of a block is related to the registry.

- A character string starting with “In subkey:”.
- A character string starting with “Sets value:”.
- A character string starting with “With data:”.

Since the synonym notation list introduced in Sect. 3.2 covers all prefixes of the registry operation description, we can use these prefixes to identify whether the description relates to a registry operation. The description of a registry operation comprises the registry key, entry name, and entry value. As shown in Table 4, there are multiple description methods for registry operations.

The sequence of registry key, entry name, and entry value appearing in vendor reports can be divided into cases that describe ‘Y’ pairs of entry key names and values after ‘X’ number of registry keys, and cases that describe ‘X’ pairs of entry names and values followed by ‘Y’ number of registry keys. In addition, as this sequence can appear in the reverse order, there are four description methods in total. The above X and Y are integers greater than 0. The regular expression used to extract the description method is as follows.

```
(s+(vd)+|s+(dv)+|(dv)+s+|(vd)+s+),
```

where `s` denotes the registry key, `v` denotes the entry name, and `d` denotes entry value, respectively.

Based on the above rules, the procedure of registry extraction is 1, 7, 3~6 described in Table 1.

3.3.3 Network Operation Extraction

The following rules are applied to determine whether each element of the block is a network operation.

- The first element delimited by “/” is a character string of the IP address.
- The first element delimited by “/” is a character string ending with a top level domain or “.”.

We also use regular expression to determine whether a character string is an IP address. In accordance with the above rules, the procedure of network extraction is 1, 8, 3, 9, 5, 6 described in Table 1.

3.3.4 Mutex Operation Extraction

Microsoft’s analysis reports do not use specific phrases, such as “In subkey:” and “Sets value” in the registry to point out a mutex. Furthermore, the mutex does not start with fixed prefixes, such as the file path (%windir%, %appdata%), registry (hklm, hkcr), or URL schema (http://, https://). Therefore, if the character string “mutex” appears in the description of a malicious behavior, we consider that description to be related to a mutex. The extraction methods for malicious behavior used in a file, registry, and network are not suitable for a mutex; thus, we store the element of the block to the database without any processing. We save the list of behavior descriptions and malware names in the same manner as the other three operations.

3.3.5 Malicious Behavior Database

We store four types of extracted operation mentioned before into Malicious Behavior Database which is built in the JSON format. The content of Malicious Behavior Database is shown in Table 3. To generate the type of malware, we not only extract behavior and description, but also malware type from vendor reports.

3.4 Malicious Behavior Extraction from Sandbox Logs

In this subsection, we present how to extract malicious behavior from sandbox logs. Note that we only use labels provided by Microsoft. Known and unknown specimens exist in the dataset, where a "known/unknown specimen" represents a sample that was/was not marked as malware with the Microsoft anti-virus checker. However, we perform detection for all specimens. With regard to detection, the file, registry, and mutex operations are based on API calls. On the other hand, we detect a network operation by comparing the "hosts" and "domains" elements of the sandbox logs using the domain and IP address stored in the malicious behavior database. Note that we do not investigate the purpose of communication or whether the domain is considered malicious.

The malicious behavior of a registry operation can be detected both when an entry value is changed and when the current entry value is confirmed relative to a specific registry key and entry name because it is not necessary to change the entry value in an environment wherein the setting has already been changed. Here we use an API Behavior MAP to detect file, registry, and mutex operations. The category and API calls of the API behavior MAP are shown in Table 5. When detecting malicious behavior, we first specify the API calls extracted from the sandbox logs using the API Behavior MAP and then query the malicious behavior database using the selected API calls. All necessary information for the database query exists in each category of the API Behavior MAP. All necessary information is listed in Table 6. Most of the information described in the Table 6 comprises arguments of the API calls. As an exception, only the registry keys category does not require arguments, such as reg_set, reg_query, reg_query_key. We process the API calls of the registry operation as follows.

1. Open the registry key specified by the category reg_open (reg_create)
2. Process the handle, entry name, and entry value related to reg_set (reg_query, reg_query_key)
3. Close the registry key specified by the close_handle category.

Only reg_open and reg_create are explicitly provided with arguments. We extract other API calls of a registry operation using the handle obtained from reg_open and reg_create. Moreover, reg_open and reg_create are sometimes used as nested elements. In this case, the full path can be obtained by concatenating each category the of registry key. By preserving the full path and handle in the malicious behavior database, we can identify the registry key required by reg_set, reg_query, reg_query_key.

Table 5 API Behavior MAP.

Category	API
close_handle	`RegCloseKey, NtClose`
reg_open	`RegOpenKeyExA, RegOpenKeyExW, NtOpenKey, NtOpenKeyEx`
reg_create	`RegCreateKeyExA, RegCreateKeyExW, NtCreateKey`
reg_enumerate	`NtEnumerateKey`
reg_set	`NtSetValueKey, RegSetValueExA, RegSetValueExW,`
reg_query_key	`NtQueryKey`
reg_query	`RegQueryValueExA, RegQueryValueExW, RegQueryInfoKeyExA, RegQueryInfoKeyExW, NtQueryValueKey`
reg_del	`RegDeleteKeyA, RegDeleteKeyW, NtDeleteKey, RegDeleteValueA, RegDeleteValueW, NtDeleteValueKey`
open_file	`NtOpenFile`
create_file	`NtCreateFile, CreateFile, CreateFileA, CreateFileW, CreateFile2,CreateFileTransacted`
copy_file	`CopyFile, CopyFile2, CopyFileA, CopyFileW, CopyFileExA, CopyFileExW`
create_dir	`CreateDirectoryExW, CreateDirectoryA, CreateDirectoryW, CreateDirectoryEx, CreateDirectoryExA, CreateDirectory`
mutex	`NtCreateMutant, NtOpenMutant`

Table 6 Necessary Information for the detection of malicious behaviors.

Category	Necessary Information
copy_file	copied file name
open_file	opened file name
create_file	created file name
create_dir	directory name
reg_open	registry key
reg_create	registry key
reg_set	key, entry name, entry value
reg_query	key, entry name
mutex	mutex name
Others	Not detected

3.5 Malware Analysis Report Generation

In this subsection, we describe a method to automatically generate a new malware analysis report based on summarizing the detected malicious behaviors described in Sect. 3.4. The malware analysis report includes the file name and type name of the detected malware, all malicious behaviors related to the detected malware, a list of actually detected malicious behaviors, the name of malware with the same detected malicious behaviors, and a list of the regular expressions used to detect those malicious behaviors.

The report generation procedure is described as follows.

1. Extract and obtain the type name of Microsoft's specimen from FFRI datasets.

2. Summarize malicious behaviors using dictionary objects. The key is the regular expression and the value is the malicious behavior matched with the corresponding regular expression.
3. Select the most suitable descriptive text in each regular expression.
4. List the type name of the malware with the same detected malicious behavior.
5. If the malware specimen is labeled, list all malicious behaviors of that specimen.

The rules for the above selection method are described as follows.

- If the specimen is known, select a description extracted from its own type.
- If there are multiple descriptions in one specimen, select the one with the fewest characters.
- If there is no description extracted from its own type, select the one with fewest characters from similar specimens. Note that the descriptions extracted for each category have similar meanings. To improve the reading speed of generated reports, we selected the description with the fewest characters.

4. Evaluation

In this section, we first present that our methodology can automatically extract malicious behaviors from vendor reports. We then show that the AMAR-Generator system can detect corresponding malicious behaviors from the sandbox logs with high detection rates. Finally, we demonstrate that the AMAR-Generator system can generate human-readable malware analysis reports.

4.1 Malicious Behaviors Extracted from Vendor Reports

Using the techniques described in Sect. 3.3, we present how the AMAR-Generator system extracts the malicious behaviors from vendor reports. We extracted 9,177 malicious behaviors from 1,678 vendor reports and stored these malicious behaviors in our database. Our extraction approach takes advantage of the information described in the vendor reports. We randomly selected 100 extracted behaviors from each type of operation for manual inspection. If the extracted information is consistent with its regular expression, we label it as correct. We confirmed that all the sampled behaviors were correctly extracted. The number of malicious behaviors in each operation is listed in Table 7.

The numbers of file and network operations are 4,416 and 3,505, respectively, and these operations account for approximately 86% of the extracted malicious behaviors. Note that the abuse of file and network operations is a major modus operandi of malware. Malware must expand infection by copying itself or refreshing itself by downloading a new version over the Internet. Registry operations are commonly used because malware attempts to hide itself to evade identification.

Table 7 Number of extracted malicious behaviors.

Type of malicious behaviors	Number of extracted malicious behavior
File operation	4,416
Registry operation	1,100
Network operation	3,505
Mutex operation	156

Table 8 Number of detected malicious behaviors in each FFRI Dataset.

Data set	Min.	Max.	Ave.	Percentage of logs detected with malicious behaviors
FFRI Dataset 2013	0	63	9.53	96.74%
FFRI Dataset 2014	1	875	30.07	100%
FFRI Dataset 2015	0	320	4.64	74.87%

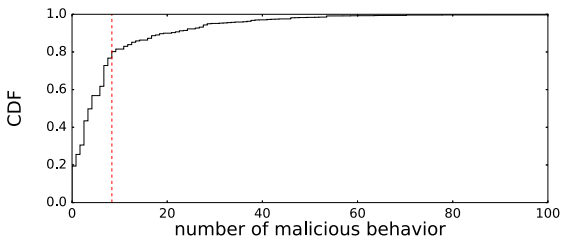

Fig. 2 The CDF of number of malicious behaviors included in one specimen ($0 \leq x \leq 100$).

4.2 Malicious Behaviors Detected from Sandbox Logs

Using the techniques described in Sect. 3.4, we present how the AMAR-Generator system extracts the malicious behaviors from sandbox logs. We used 8,640 sandbox logs as input to evaluate the detection performance of the proposed system.

Table 8 shows the number of detected malicious behaviors in each FFRI dataset. As can be seen, the minimum, maximum, and average numbers of detected malicious behaviors are 0, 875, and 14.96, respectively, in these three datasets. The proposed system can detect all malicious behavior in the sandbox logs without bias. Moreover, our system achieves very high detection rates of malicious behaviors reported in the vendor reports; i.e., 96.74% and 100% with the 2013 and 2014 FFRI datasets, respectively. Note that the detection rate conveys the coverage ratio of sandbox logs in our proposed system. On the other hand, the proposed system can identify malicious behaviors in most of the sandbox logs from the 2015 FFRI dataset. We leverage the knowledge in vendor reports to specify malicious behaviors, as all extracted malicious behaviors are described in vendor reports. Thus, the false positive rate is 0. This proves that our system is effective for discovering malicious behavior from sandbox logs.

Figure 2 shows the Cumulative Distribution Function (CDF) of the number of malicious behaviors that can be detected from each specimen. The red dotted line in Fig. 2 expresses the average number of malicious behaviors. By

Table 9 Top 10 malware types undetected in FFRI Dataset 2015.

Malware type	Number of specimens (Percentage)	Number of behavior extracted from vendor reports	Only exists in FFRI Dataset 2015
unknown	259 (34.35)	0	unknown
virtool:win32/ceeinject.gen!kk	103 (13.66)	0	False
virtool:win32/obfuscator.wt	45 (5.97)	0	False
trojan:win32/dynamer!ac	25 (3.32)	0	False
trojandownloader:win32/small.gen!i	20 (2.65)	0	True
trojandropper:win32/bunitu.c	15 (1.99)	33	True
backdoor:win32/kelihos!rfn	14 (1.86)	0	True
trojan:win32/ropest.j	13 (1.72)	0	True
vrogue:win32/trapwot	11 (1.46)	7	True
backdoor:win32/kelihos	10 (1.33)	7	True

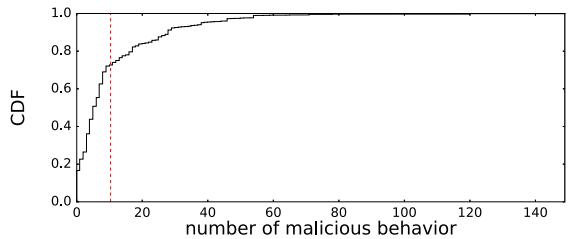

Fig. 3 CDF of number of malicious behaviors in unknown type of logs.

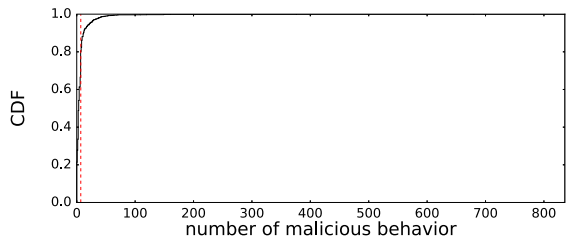

Fig. 4 CDF of number of malicious behaviors in known type of logs.

```
Malware name is "worm:win32/nuqel.h"

This malware has following behaviors

Upon execution, this file drops the following copies of
itself with the read-only, system,and hidden file attributes:
· %windir%\scvhsot.exe
· %windir%\hinhem.scr
· \scvhsot.exe
· \blastclnnn.exe

this file copies itself to one of the following folders with
read-only, hidden and system attributes:
· %systemroot%\system32

It then copies itself in the root of the found shared drives
as the following files:
· new folder.exe
· scvhsot.exe

It modifies the system registry so that it runs every time
Windows starts:
· hkcu\software\microsoft\windows\currentversion\run\yahoo
messenger\\scvhsot.exe
· hklm\software\microsoft\windows nt\currentversion\winlogon
\shell\explorer.exe scvhsot.exe
```

Fig. 5 An excerpt of generated report for "Worm:Win32/Nuqel.H".

comparing the CDF of the number of malicious behaviors to the minimum number of detected malicious behaviors, we observed that the malicious behaviors cannot be discovered from 20% of the sandbox logs. It turns out that most of these logs belong to the 2015 FFRI dataset.

Figures 3 and 4 show the CDF of the number of malicious behaviors in unknown and known specimen logs, respectively. The red dotted lines in Fig. 3 and 4 express the average number of malicious behaviors. We found that malicious behaviors were detected from 83.37% unknown specimen logs and 78.56% known specimen logs. Note that there are existing malicious behaviors in unknown specimen logs, and the proposed system is effective for detecting malicious behaviors from such logs.

To provide more insight about uncovered sandbox logs, we summarized the type of malware that could not be detected in the 2015 FFRI dataset. As shown in Table 9, most of the malicious behaviors of malware could be extracted from the vendor reports and most malware types only exist in the 2015 FFRI dataset. Note that many undetected specimens in the 2015 FFRI dataset are new malware; therefore, the malicious behaviors related to such malware are not described in the vendor reports. We also consider that dynamic analysis systems, such as cuckoo, may not capture malicious behaviors due to the evasive nature of some malware.

4.3 Auto-generated Malware Analysis Report

Using the techniques described in Sect. 3.5, we demonstrate how the AMAR-Generator system generates human-readable reports from the sandbox logs. As an example, we make use a pair of a vendor report and a sandbox log. Both the report and the log are associated with a type of malware named "Worm:Win32/Nuqel.H." Note that the environments used to obtain the vendor report and the sandbox log are different, indicating that there should be intrinsic difference between the two information sources. Nevertheless, we see the our approach can successfully generate a report that contains similar information found in the vendor report.

Figures 5 and 6 present the generated report and the vendor report. Due to the space limitation and conciseness, we show only excerpts of the reports, containing file operation and registry operation. Other operations such as network and mutex are omitted. We notice that the description of malicious behaviors in the auto-generated report is concise and human-readable like the vendor report. Several

```
Threat behavior

Worm:Win32/Nuqel.H is a worm that spreads via removable and
shared drives.

Installation
Upon execution, Worm:Win32/Nuqel.H drops the following copies
of itself with the read-only, system, and hidden file
attributes:
· <system folder>\scvhsot.exe
· <system folder>\blastclnnn.exe
· %windir%\hinhem.scr
· %windir%\scvhsot.exe

Removable Drives
Worm:Win32/Nuqel.H copies itself in the root of the found
shared drives as the following files:
· New Folder.exe
· scvhsot.exe

It modifies the system registry so that it runs every time
Windows starts:

Modifies value: "Shell"
With data: "explorer.exe scvhsot.exe"
To subkey: HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion
\Winlogon

Adds value: "Yahoo Messengger"
With data: "<system folder>\scvhsot.exe"
To subkey: HKCU\Software\Microsoft\Windows\CurrentVersion\Run

It also drops the file "autorun.ini" in the Windows system
folder,which enables this worm to run every time a folder is
automatically opened (for example, when a user inserts a
removable disk or a CD).
```

Fig. 6 An excerpt of vendor report for "Worm:Win32/Nuqel.H".

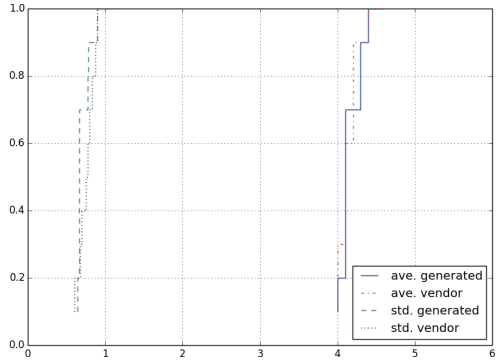

Fig. 7 CDF of average score and standard deviation of readability.

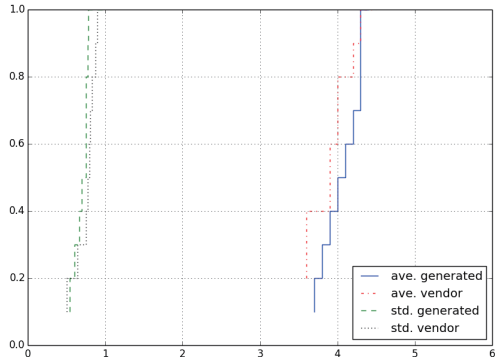

Fig. 8 CDF of average score and standard deviation of helpfulness.

common malicious behaviors, i.e., copying itself, dropping files can be found in the two reports. There are some grammatical errors in the our auto-generated reports, such as "It then" in the first sentence of description and the first lowercase character of the sentence. Although such errors may not have the effect on the understanding of our reports, we can easily fix them by introducing simple heuristics. We also notice that the auto-generated report did not provide the additional description like vendor reports; i.e, the description in the last part of the vendor report shown in Fig. 6. We leave adding those additional descriptions for our future work.

4.4 Readability and Helpfulness

The design's goal is to evaluate and compare the readability and helpfulness of our reports with vendor reports. Herein, we perform a user study on our generated reports, described in Sect. 4.3, using Google Forms [9]. The user study questionnaire includes two parts: scale-based questions and a written response.

With respect to the scale-based questions, we randomly select 10 of our generated reports and 10 vendor reports, and then shuffle these two types of reports together in random order. For each report, we ask the following two questions:

- Do you think this report is readable?
- How helpful is this report for malware analysis?

We use a 5-scale rating for answering these two questions, where a rating of 5 means the best readability or helpfulness and a rating of 1 means the worst readability or helpfulness. At the end of the questionnaire, we ask participants to answer the following written response: "*Please briefly state how these reports are useful for malware analysis.*"

We hired 10 participants who have at least two years of experience in malware analysis to answer our questionnaire. Note that we randomly sort the order of our reports and vendor reports, and do not specify which reports are generated, so that the participants can evaluate these two types of reports without bias.

Figures 7 and 8 present the CDF of the average score and standard deviation of readability and helpfulness. For the scale-based question, we calculated the average score and standard deviation of our generated reports and vendor reports, separately. We observed that slight differences between the average score of our generated reports and vendor reports; however, the standard deviation of both types of reports is less than 1.0. Such results indicate that our generated reports are extremely close to vendor reports in terms of readability and helpfulness, thus the analyst can use our generated reports for malware analysis instead of vendor reports. Additionally, we summarize the answers of 10 participants' questionnaire. The participants claimed that when conducting static code analysis, they could quickly and easily locate the analysis part in malware samples with the hints and features of our reports. Moreover, they were able to build an automatic detection system using the behav-

iors stated in the reports.

5. Related Work

Many analysis methods related to our system have been proposed in recent years. Such methods can be classified into two categories depending on the research objective. In this section, we review related work relative to these two categories and compare them to the proposed system.

Malware detection and classification

In [9], Ahmed et al. combined a time series and the inputs and outputs of API calls as classifier features, proving that their features can contribute to malware detection in this supervised machine learning method. Previous work [10], [11] treated the information of API calls obtained from the IDA Pro disassembler as natural language, then applied an n-gram as feature extraction approach for detecting malware samples. Nakazato et al. [12] proposed a new classification approach that can achieve effective and efficient classification of malware using a sequence of Windows API calls captured by a micro analysis system. Bayer et al. [13] developed an unsupervised machine learning technique for clustering malware samples based on the behaviors gleaned from dynamic analysis logs in a scalable manner. Furthermore, Li et al. [14] evaluated and compared the performance of previous methods related to the use of cluster analysis, including the work of Bayer et al. Their finding was that the ground truth of the analysis data has a significant effect on cluster analysis accuracy. Inoue et al. [15] proposed a method to fully automate large-scale malware analysis in a virtual Internet environment. Their system can hook API calls by overwriting the Import Address Table. Mohaisen et al. [16] compared the analysis results of different antivirus vendors to that of their own system to evaluate the correctness of the antivirus vendors' analysis results.

Our system also treats sandbox logs as input, but we aim to automatically interpret the API calls and parameters for security analysts, based on the knowledge described in the vendor reports. We believe that the proposed system can serve as an important complement to previous research outcomes regarding the analysis and detection of malware.

Text Generation

Zhang et al. [17] developed a novel approach that can use the result of static program analysis to automatically produce security-centric android application descriptions. Yu et al. [18] designed AutoPPG to automatically generate correct and readable privacy policy for Android applications by applying natural language processing techniques on the result of static code analysis.

In our study, we have attempted to automatically generate human-readable malware analysis reports by leveraging the results of dynamic program analyses. Our system produced the description of Windows malware, which had not been previously done. Our goal is to improve the effectiveness and efficiency of malware analysis for security analysts.

6. Discussion

In this section, we discuss the limitations of our work.

Mis-detection of Malicious Behaviors

We assume that all the behaviors saved in the malicious behavior database are in fact malicious. However, some behaviors are stored in the form of regular expressions; thus, it is possible that some normal behaviors are detected as malicious behaviors. The purpose of our work is to provide all possible useful information, including suspicious information, to security analysts in the form of human-readable reports and accelerate the malware analysis process as much as possible. A security analyst can refer to our report for further malware analysis, such as static analysis. Through further malware analysis, the security analysts may find such misdetections and previously unknown malicious behaviors from suspicious behaviors.

Type of malicious behavior

In this paper, we have focused on extracting four kinds of malicious behaviors, i.e., file, registry, network, and mutex operations. Since the description of these operations has a specific format in the vendor reports, it is possible to extract the malicious behavior of such operations. However, there are other types of behaviors that are abused by malware in the sandbox logs. For example, `SetWindowsHookEx` and `CreateRemoteThread` can be used to accomplish code injection, and the proposed system cannot identify such malicious behavior at this point. We think that this limitation can be resolved using analysis reports created by other antivirus vendors because the formats of the analysis reports created by each vendor differ, and other types of malicious behaviors may be described in a specific format. We leave the challenge of expanding the different types of malicious behavior the proposed system can handle as the focus of future work.

Dependence on cuckoo sandbox

As mentioned in Sect. 2, the input to the proposed system (i.e., sandbox logs) is the analysis result of the cuckoo sandbox, and the number of malicious behaviors captured by the cuckoo sandbox has an impact on the performance of the system. There are many factors that can lead to this limitation. For example, the malware can use a timer to trigger malicious behaviors after a maximum execution time set by cuckoo sandbox, or some types of malicious behaviors can only be captured in a specific operating system. Thus, it is very difficult to obtain all malicious behaviors using only one combination of settings. However, we believe that our generated report is valuable feedback for the analysis results of the cuckoo sandbox, and a security analyst can adjust the cuckoo sandbox settings by referring to our report.

7. Conclusion

We proposed the AMAR-Generator system, which can automatically generate malicious behavior-related reports using reports provided by antivirus vendors for security ana-

lysts. The system comprises of several techniques, template matching, API-behavior mapping, and malicious-behavior database. Our experimental results demonstrated that it can generate malware analysis report that helps a security analyst as the first step to analyze a given malware. We also demonstrated that the proposed method achieved high detection rates of malicious behaviors from sandbox logs; i.e., detection rates up to 96.74%, 100%, and 74.87% on the 2013, 2014, and 2015 FFRI datasets, respectively. We also reported that the AMAR-Generator system can effectively and efficiently identify malicious behaviors from unknown types of sandbox logs.

References

[1] "Av-test. malware." http://www.av-test.org/en/statistics/malware.
[2] FFRI, Inc. http://www.ffri.jp/en/company/index.htm.
[3] M.D. Kamizono Masaki et al. http://www.iwsec.org/mws/2015/.
[4] "Cuckoo sandbox." https://www.cuckoosandbox.org.
[5] "Virustotal- free online virus, malware and url scanner." https://www.virustotal.com.
[6] Microsoft Malware Protection Center, "Common folder variables." https://www.microsoft.com/security/portal/mmpc/shared/variables.aspx.
[7] Microsoft, "Recognized environment variables." https://technet.microsoft.com/ja-jp/library/cc749104(v=ws.10).aspx.
[8] File-Extensions.org, "File-Extensions.org - File Extension Library." http://www.file-extensions.org/.
[9] F. Ahmed, H. Hameed, M.Z. Shafiq, and M. Farooq, "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, AISec 2009.
[10] C. Ravi and R. Manoharan, "Malware detection using windows api sequence and machine learning," International Journal of Computer Applications, vol.43, no.17, pp.12–16, 2012.
[11] M. Alazab, S. Venkataraman, and P. Watters, "Towards understanding malware behaviour by the extraction of API calls," 2010 Second Cybercrime and Trustworthy Computing Workshop, pp.52–59, 2010.
[12] J. Nakazato, J. Song, M. Eto, D. Inoue, and K. Nakao, "A novel malware clustering method using frequency of function call traces in parallel threads," IEICE Trans. Inf. & Syst., vol.E94-D, no.11, pp.2150–2158, 2011.
[13] U. Bayer, P.M. Comparetti, C. Hlauschek, C. Krügel, and E. Kirda, "Scalable, behavior-based malware clustering," Proc. Network and Distributed System Security Symposium, NDSS, 2009.
[14] P. Li, L. Liu, D. Gao, and M.K. Reiter, "On challenges in evaluating malware clustering," Recent Advances in Intrusion Detection, Lecture Notes in Computer Science, vol.6307, pp.238–255, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.
[15] D. Inoue, K. Yoshioka, M. Eto, Y. Hoshizawa, and K. Nakao, "Automated malware analysis system and its sandbox for revealing malware's internal and external activities," IEICE Trans. Inf. & Syst., vol.E92-D, no.5, pp.945–954, 2009.
[16] A. Mohaisen and O. Alrawi, "Av-meter: An evaluation of antivirus scans and labels," Detection of Intrusions and Malware, and Vulnerability Assessment, Lecture Notes in Computer Science, vol.8550, pp.112–131, Springer International Publishing, Cham, 2014.
[17] M. Zhang, Y. Duan, Q. Feng, and H. Yin, "Towards automatic generation of security-centric descriptions for android apps," Proc. 22nd ACM SIGSAC Conference on Computer and Communications Security- CCS '15, pp.518–529, 2015.
[18] L. Yu, T. Zhang, X. Luo, L. Xue, and H. Chang, "Toward automatically generating privacy policy for android apps," IEEE Trans. Inf. Forensics Security, vol.12, no.4, pp.865–880, 2017.

Bo Sun received B.E degree in computer science from Jilin University in 2007, M.E degree in Information and Media from Yokohama National University in 2012, and Ph.D degree in the Department of Computer Science and Engineering from Waseda University in 2018. He was a research associate in the Department of Computer Science and Engineering, Waseda University from 2016 to 2018. He is currently a researcher with the National Institute of Information and Communications Technology. His research interest is network security and mobile security.

Akinori Fujino was born in 1991. He received B.E. and M.E degrees in computer science and engineering from Waseda University in 2014 and 2016, respectively. He has been engaged in the research of cyber security.

Tatsuya Mori is currently a professor at Waseda University, Tokyo, Japan. He received B.E. and M.E. degrees in applied physics, and Ph.D. degree in information science from the Waseda University, in 1997, 1999 and 2005, respectively. He joined NTT lab in 1999. Since then, he has been engaged in the research of measurement and analysis of networks and cyber security. From Mar 2007 to Mar 2008, he was a visiting researcher at the University of Wisconsin-Madison. From May 2018, he is a visiting researcher at the center for advanced intelligence project (AIP), RIKEN. He received Telecom System Technology Award from TAF in 2010 and Best Paper Awards from IEICE and IEEE/ACM COMSNETS in 2009 and 2010, respectively. Dr. Mori is a member of ACM, IEEE, IEICE, IPSJ, and USENIX.

Tao Ban is currently a senior researcher with Cybersecurity Research Institute, National Institute of Information and Communications Technology, Tokyo, Japan. He got his Bachelor's degree from Department of Automatic Control, Xi'an Jiaotong University in 1999, his Master degree of engineering from Department of Automation, Tsinghua University in 2003, and Ph.D. from Kobe University in the field of Information Science in 2006. His research interest includes cybersecurity,neural processing, machine learning, and data mining.

Takeshi Takahashi received the Ph.D. degree in telecommunication from Waseda University, in 2005. He was with the Tampere University of Technology as a Researcher from 2002 to 2004, and Roland Berger Ltd., as a Business Consultant, from 2005 to 2009. Since 2009, he has been with the National Institute of Information and Communications Technology, where he is currently a Research Manager. His research interests include Internet security and network protocols.

Daisuke Inoue received his B.E. and M.E. degrees in electrical and computer engineering, and Ph.D. degree in engineering from Yokohama National University in 1998, 2000 and 2003, respectively. He joined Communications Research Laboratory (CRL), Japan, in 2003. CRL was relaunched as National Institute of Information and Communications Technology (NICT) in 2004, where he is currently the director of Cybersecurity Laboratory. He received several awards including the best paper award at the 2002 Symposium on Cryptography and Information Security (SCIS 2002), the commendation for science and technology by the minister of MEXT, Japan, in 2009, the Good Design Award 2013, the Asia-Pacific Information Security Leadership Achievements 2014, and the award for contribution to Industry-Academia-Government Collaboration by the minister of MIC, Japan, in 2016.