Poster:Toward Automating the Generation of Malware Analysis Reports Using the Sandbox Logs

Bo Sun
Waseda University
3-4-1 Okubo Shinjuku
Tokyo, Japan 169-8555
sunshine@nsl.cs.waseda.ac.jp

Akinori Fujino
Waseda University
3-4-1 Okubo Shinjuku
Tokyo, Japan 169-8555
fujino@nsl.cs.waseda.ac.jp

Tatsuya Mori
Waseda University
3-4-1 Okubo Shinjuku
Tokyo, Japan 169-8555
mori@nsl.cs.waseda.ac.jp

ABSTRACT

In recent years, the number of new examples of malware has continued to increase. To create effective countermeasures, security specialists often must manually inspect vast sandbox logs produced by the dynamic analysis method. Conversely, antivirus vendors usually publish malware analysis reports on their website. Because malware analysis reports and sandbox logs do not have direct connections, when analyzing sandbox logs, security specialists cannot benefit from the information described in such expert reports. To address this issue, we developed a system called *ReGenerator* that automates the generation of reports related to sandbox logs by making use of existing reports published by antivirus vendors. Our system combines several techniques, including the Jaccard similarity, Natural Language Processing (NLP), and Generation (NLG), to produce concise human-readable reports describing malicious behavior for security specialists.

Keywords

Malware Analysis, Sandbox Logs, Reports, Natural Language Processing

1. INTRODUCTION

Computer malware remains a significant threat to our daily lives. According to a report by AV-TEST [1], nearly 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 for understanding the various features of malware, after which malware detection systems can be developed and improved upon. Malware analysis can generally be divided into two categories, i.e., static and dynamic. The static method is incapable of extracting such features of malware as obfuscation code, whereas the dynamic method can leverage the actual controlled environment so as to detect malicious behavior hidden by obfuscation code.

For security specialists, the dynamic method is indispensable for analyzing examples of malware. These experts investigate sandbox logs, which are the output of the dynamic method, including the cuckoo sandbox [2], for identifying malicious behavior; however, it is inefficient to analyze such large numbers of sandbox logs via manual inspection. As such, antivirus vendors analyze huge volumes of malware. Their analysis reports are stored in a database and are typically open to public access via the Internet. In general, malware analysis reports are written in natural languages and do not include details of various API calls or related arguments. Further, malware analysis reports are relatively independent of one another in terms of malware types, thus it is difficult to associate sandbox logs with these reports in terms of API calls and corresponding arguments; more generally, it is difficult to extract either the same or different characteristics from these reports. When analyzing sandbox logs, security specialists cannot effectively or efficiently retrieve useful information described in expert reports. because there are a vast number of expert reports exsited in real world and many expert reports are generated constantly.

To bridge this gap, we developed a system called *ReGenerator* to automatically generate malware analysis reports that precisely describe the malicious behavior found in sandbox logs based on knowledge presented in expert malware analysis reports. To interpret the behavior described in these two types of documents, we first obtain API calls and the value name from sandbox logs, using these behaviors as searching data. Second, we replace abstract expressions, such as "<random number>.exe," in expert reports by applying regular expressions. To create a relative correlation, we adopt the Jaccard similarity to compute the similarity score between searching data and expert reports. Based on similar part in expert reports and the MSDN API [7], we leverage NLP and Natural Language Generation (NLG) to produce concise human-readable reports describing malicious behaviors.

The chief contributions of our work are summarized as follows.

- To the best of our knowledge, this is the first work that attempts to automate the generation of malware analysis reports to accelerate the malware analysis process. Our system is capable of serving as an assistant tool to alleviate the burden of malware analysis for security specialists.
- We propose a system called *ReGenerator* that automatically presents malicious behaviors described in sandbox logs to security specialists using natural language.
- To generate a concise human-readable report describing malicious behavior, we applied an efficient and effective combination of the Jaccard similarity, NLP, and NLG.

In addition to this introductory section, the remainder of our paper is organized as follows. In Section 2, we present the dataset used by ReGenerator. Next, we describe the ReGenerator in Section 3. In Section 4, we summarize related work and compare

CCS'16 October 24-28, 2016, Vienna, Austria
© 2016 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-4139-4/16/10.
DOI: http://dx.doi.org/10.1145/2976749.2989064

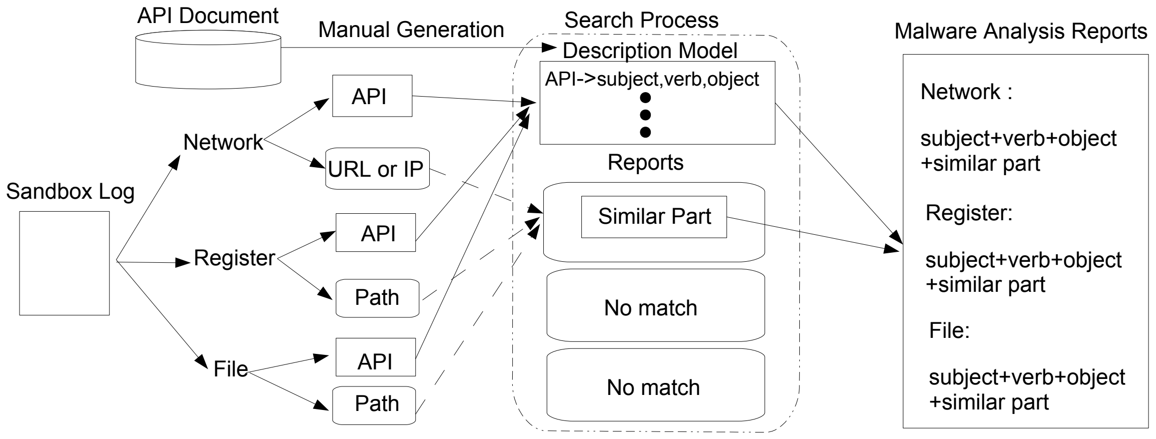

Figure 1: Overview of the ReGenerator System.

such work with our own. Finally, we present our conclusions and avenues for future work in Section 5.

2. DATASET

In this section, we describe the sandbox logs that we used and how we gathered expert malware analysis reports from the “real world.”

Sandbox Logs We used the FFRI dataset from 2013 to 2015 provided by FFRI, Inc. [11]. The FFRI dataset is also part of the MWS datasets [12] collected by different research institutes and industries in Japan. Note that the number of malware samples was 8,644. FFRI dataset content was represented as JSON files that were the output of the cuckoo sandbox [2]. 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 [6], 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, parameter, and category.

Expert's Malware Analysis Report Many antivirus vendors, such as Microsoft and Symantec, publish malware analysis reports on their website. We constructed a crawler to collect the HTML 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 from the FFRI datasets, then 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 are different for different antivirus vendors, our system was designed to cope with different antivirus vendor reports. In this paper, we only present information from Microsoft's reports as an example. Microsoft's reports consisted of the following four parts: “Summary”; “what to do now”; “Technical information”; and “Symptoms.” The “Technical information” portion presented details of the malicious behaviors, so we only considered that portion of Microsoft's reports.

3. REGENERATOR SYSTEM

In this section, we present the architecture of our ReGenerator system (see fig. 1). We extract useful information from a sandbox log. We locate network traffic, and registry and file information created or accessed by malware samples from the sandbox logs by specifying such JSON keys as api, host, and domain and then we obtain API call and value name such as URL, IP, Path as pairs with regard to these information. Note that we only focus on extracting API calls with the value name like URL, IP and Path, because these kind of API calls is most likely related to malicious behaviors.

At the same time, because the function name of API calls is not presented in expert reports, we manually created a description model to address this issue by referring to MSDN API documentation. The output of our model here is a 3-tuple of natural language words consisting of subject, verb, and object. For example, the Windows API call ZwMapViewOfSection is documented by Microsoft as follows: “The ZwMapViewOfSection routine maps a view of a section into the virtual address space of a subject process.” We built our model by manually mapping this function name from the API into “The malware,” “map,” and “section” much like a reference table. When this API function is called, our model automatically confirms the reference table, then assigns the mapped result to our generated report.

After extracting useful information and building the description model, we connect Sandbox log's information with expert reports by implementing search process. Expert reports do not contain function names of API calls, thus for the function name of API calls and the value name, we apply description model and expert reports to complete the search process, respectively. With regard to the function name of API calls, we use description model to complete the translation task and then generate more readable description by using NLG engine [4]. The input of NLG engine is a 3-tuple of natural language words.

As for the value name, we discover same or similar description in expert reports by using similarity matching. Before string match, as data preprocessing, we leveraged Block-Level Elements [15] and Inline Elements [16] to convert the HTML pages of the “Technical Information” part to plaintext, then applied regular expressions to replace abstract expressions, such as “<random number>.exe” and “HKLM\SOFTWARE\<8-digit hexadecimal number>” in the expert reports.

To build the feature vector for similarity matching, we first implemented the following procedure to clean up the expert reports and the value name.

Step1: Split all sentences into words.

Step2: Transform all letters to lowercase.

Step3: Remove all stop words such as "is","am","the". etc.

Step4: Consolidate variant forms of a word into a common form using a word stemming algorithm (e.g., converting " running " to " run ").

Step5: Correct misspelled English words in all reports.

We implemented NLP based on NLTK [3] and TextBlob [5]. NLTK is a widely used Python library for NLP, and TextBlob was developed based on NLTK to simplify text processing. TextBlob enabled us to easily implement language detection and spelling correction functionality. After cleaning up the value name and expert reports, we used the " bi-gram " approach to create a feature vector for discovering similar string patterns. We measured the similarity between the value name and the expert reports by using Jaccard Similarity. The threshold of Jaccard Similarity is 0.9. If description in expert report is similar with the value name, we record similar part that can be determined by information retrieval to malware analysis report. When writing translated API calls and similar part to malware analysis report, we combine these two things to sentence according to the relation of pairs we extract in the first stage.

4. RELATED WORK

Many analysis methods related to our system have been proposed in recent years. Such methods can be classified into three categories depending on the approach used. In this section, we review related work from these three categories and compare them with ours.

Supervised Machine Learning Ahmed et al. combines time series and the values of input and output in API calls as classifier's features and proved that their features can contribute to malware detection in machine learning method. Previous works [17, 9] treated the infromation of API calls obtained from IDA pro disassembler as natural language and applied n-gram as feature extraction approach to detect malware samples.

In [8], 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 [17, 9] 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.

Unsupervised Machine Learning In [10], Bayer et al. proposed a novel unsupervised machine learning technique for clustering malware samples based on behaviors gleaned from dynamic analysis logs in a scalable way. Further, in [13], Li et al. evaluated and compared the performance of previous works related to the use of cluster analysis, including the work of Bayer et al. Their finding was that the ground truth of analysis data has a significant effect on the accuracy of cluster analysis.

Non-Machine Learning In [14], Mohaisen et al. compared analysis results of each antivirus vendor with that of their own system to evaluate the correctness of antivirus vendor analysis results.

In our study, we attempt to address this problem using a novel approach that automatically generates malicious behavior-related reports to improve the effectiveness and efficiency of malware analysis for security specialists. We believe that our system will serve as an important complement to past research outcomes on the analysis and detection of malware samples.

5. CONCLUSION AND FUTURE WORK

In this study, we proposed the ReGenerator system, which can automatically generate malicious behavior-related reports using expert reports provided by antivirus vendors for security specialists. In our future work, we plan to extend our crawler to collect more types of expert reports, such as those produced by Transcend. Further, we plan to perform a user study to evaluate the readability and effectiveness of our auto-generated reports. We plan to implement our user study based on two key goals. First, we expect to figure out whether the auto-generated reports are readable and easily understood by average security specialists. Second, we aim to determine whether our reports can actually help security specialists quickly and correctly analyze malware logs.

Acknowledgements

A part of this work was supported by JSPS Grant-in-Aid for Scientific Research B, Grant Number JP16H02832.

6. REFERENCES

[1] Av-test. malware. http://www.av-test.org/en/statistics/malware.

[2] Cuckoo sandbox. https://www.cuckoosandbox.org.

[3] Natural language toolkit. http://www.nltk.org.

[4] simplenlg: Java api for natural language generation. https://code.google.com/p/simplenlg/.

[5] Textblob: Simplified text processing. http://textblob.readthedocs.io/en/dev/.

[6] Virustotal- free online virus, malware and url scanner. https://www.virustotal.com.

[7] Windows api index. https://msdn.microsoft.com/ja-jp/library/windows/desktop/ff818516.aspx.

[8] F. Ahmed, H. Hameed, M. Z. Shafiq, and M. Farooq. Using spatio-temporal information in API calls with machine learning algorithms for malware detection. In *Proceedings of the 2nd ACM Workshop on Security and Artificial Intelligence, AISec 2009*.

[9] M. Alazab, S. Venkataraman, and P. Watters. Towards understanding malware behaviour by the extraction of api calls. In *n Cybercrime and Trustworthy Computing Workshop (CTC), 2010*.

[10] U. Bayer, P. M. Comparetti, C. Hlauschek, C. Krügel, and E. Kirda. Scalable, behavior-based malware clustering. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2009*.

[11] FFRI.inc. http://www.ffri.jp/en/company/index.htm.

[12] M. D. . Kamizono Masaki et al. http://www.iwsec.org/mws/2015/.

[13] P. Li, L. Liu, D. Gao, and M. K. Reiter. On challenges in evaluating malware clustering. In *Recent Advances in Intrusion Detection,RAID 2010*.

[14] A. Mohaisen and O. Alrawi. Av-meter: An evaluation of antivirus scans and labels. In *Detection of Intrusions and Malware, and Vulnerability Assessment,DIMVA 2014*.

[15] M. D. Network and individual contributors. Block-level elements. https://developer.mozilla.org/en-US/docs/Web/HTML/Block-level_elements.

[16] M. D. Network and individual contributors. Inline elements. https://developer.mozilla.org/en-US/docs/Web/HTML/Inline_elements.

[17] C. Ravi and R. Manoharan. Malware detection using windows api sequence and machine learning. In *International Journal of Computer Applications,Vol. 43, No. 17*.