Towards a Representation of Malware Execution Traces for Experts and Machine Learning

Vincent Raulin, Pierre-François Gimenez, Yufei Han, Valérie Viet Triem Tong

Towards a Representation of Malware Execution Traces for Experts and Machine Learning

Vincent Raulin*, Pierre-François Gimenez†, Yufei Han*, Valérie Viet Triem Tong†
*Inria, Univ. Rennes, IRISA, {firstname.lastname}@inria.com
†CentraleSupélec, Univ. Rennes, IRISA, {firstname.lastname}@centralesupelec.fr

***Abstract*—Dynamic analysis is a common technique to analyze the run-time behavior of software and identify malware (malicious software). Execution traces typically contain the list of system calls with their parameters, the list of accessed files, etc. Several representations have been proposed to organize these data better and help both human experts and automated tools analyze them effectively. This paper reviews these representations and identifies four research problems that the first author plans to investigate during his Ph.D.**

***Index Terms*—malware analysis, dynamic analysis, visualization, data representation**

I. Introduction

In recent years, the number of malware (**mal**icious soft**ware**) has been increasing rapidly. Every year commercial antiviruses manufacturers make reports on their annual activities, including malware evolution data, which shows a tremendous amount of new malware per month [5]. With this rising threat, adapted defenses must be organized.

Malware analysis is the domain of computer science focusing on identifying and understanding malware. The identification task consists of distinguishing malware from benign software (also called goodware), whereas the understanding task goes further and consists of studying the program's behavior. Both tasks can be approached in different ways. Static analysis is the study of the code of the malware without execution. Dynamic analysis is based on the monitoring of the software execution. Hybrid analysis can mix both techniques.

This study will focus on dynamic analysis. This analysis is generally done with a controlled environment that collect information about the execution, for example with the Cuckoo [1] sandbox which returns an execution report. This report is very rich and can contain many different types of information which is not easy to handle right away for experts of malware analysis tools (for example, Machine Learning (ML) models).

Machine learning has been successfully applied to various domains such as image and language processing and can be used in execution traces analysis. In that case, a learning algorithm is fed with the collected data as an input to learn a model and then perform a task, for example detecting whether sample is a malware or classifying a malware into a family – virus, trojan, ransomware, etc. The transformation of the execution traces of a sample into the input data of a malware analysis system can have a large impact of the model's performances. Besides, these representations are not only useful for ML techniques: they can also be used by a human expert to understand the behavior of a malware. Many representations [3], [4], [8], [10]–[12], [18], [19] of execution traces have been proposed, with various advantages and limits.

This article presents a survey of malware execution representations and points out their limitations. This brings us closer to the goal of the Ph.D. of the first author, that is to build a representation of a program behavior that is robust to evasion attacks, can encompass executions traces from various mainstream Operating Systems (OSes), be useful for an expert to manually review and reflects all malicious actions of an execution trace.

The paper is structured as follow. Section II presents a survey of malware execution representations. Section III points out several research questions related to representation and discusses them. Finally, Section IV concludes this article and presents the goal of this Ph.D. thesis.

II. Survey of Malware Execution Representations

Our work focuses on defining and extracting efficient feature representations from dynamic traces of malware samples. The first subsection focuses on the different sources for monitoring information. The second subsection summarizes proposed representations of this information.

A. Sources of information

Most works on dynamic analysis [16] collect system call traces while executing a malware sample. Indeed, they are the main medium of interaction between a program and the host OS, and their traces can provide accurate detection output of malicious payloads [7], [8], [10], [12], [14], [19]. Their parameters are also an important source of information [8], [14], [20]. Monitoring disk access, network ports, and OS activity inside a virtual machine have also been proposed [2], [3], [18]. Finally, physical properties like CPU usage [15] or CPU power consumption [6] can also be monitored. But such information does not concern interactions with the OS, which is out of scope for this study.

For expert and classifier systems to better analyze these data, one needs to organize them inside a representation. The following subsection proposes a literature review of such representations.

B. Representations of monitoring information

We identified four families of representation, depending on how they organize the information, and thus what parts of a program's behavior they highlight.

1) Representing the order of actions: A classical representation directly uses time series of system calls without parameters [12], [19]. This representation is already used in execution reports. N-grams (sequences of n system calls) or word embedding vectors (a machine-learning-based technique) [12], [19] are then extracted to conduct malware analysis. A graph structure inspired from Markov chains has also been proposed [7], [10], where each vertex denotes a system call and each edge is labelled by the probability to chain two system calls in the observed executing traces. This representation was extended by [10] to consider n-tuples of system calls. The size of this representation does not depend on the sample or the size of the execution traces.

2) Representing the actions on objects: Parameters of system calls allow to better characterize the behavior of a program. For example, one can identify whether a "open file" system call concerns the same target file as the "write into file" system call executed later. Several representations track **objects**, e.g. files, sockets, threads, etc, manipulated by system calls as parameters or return values. [8] defines a graph where each vertex is a system call and a directed edge links every pair of consecutive system calls that manipulate the same **object** in the sequence. Another work [14], [20] consists of multiple time series, one for each set of system calls manipulating the same **object**.

3) Representing the interactions between objects: Several works rely on multiple other sources of information from the interactions with the system, e.g. network traffic, contacted IPs/URLs/domains, file access (with modifications) for better understanding malware behaviors. In [3], processes, threads, files and sockets are included in the representation. Furthermore, system calls are replaced by more accurate actions. For example, a call to "open" with a file descriptor will be replaced by "open file" with the corresponding file object, thus a sequence is transformed in a **categorized** (by type of objects it handles) version. From that point, a sequence of system calls is processed for each monitored thread. This sequence is then **categorized**, and all the objects it interacts with are formatted to include practical information (i.e., the mode this file is opened with, which process spawned which one, what thread contacted which IP on what port, etc.). Sadly, we do not have much more information on how the actual graph is built.

4) Representing the interactions between processes from different programs: The work from [18] proposes a graph-structured representation based on the monitoring of multiple processes on several machines.This is a heterogeneous graph with three node types: file, process and socket. There are three kinds of edges: from a process to a socket (if that process uses this socket to communicate), from a process to a file (if that process interacts with it) and from a process to another one (if the former creates the latter). This malware representation can identify a suspicious process by comparing its behavior with benign processes. It can also be used to analyze the behavior of a malware with respect to the files it handles, the processes it spawns and the IP address it contacts. For example, it could help identify a remote C&C (Command and Control) server.

Finally, VirusTotal's database exploration graph [2] is a representation that entails the interactions between a malware and other entities (file, IP, domains, etc.), both via internet and via the file system.

III. Discussions on Research Problems

The previous research efforts try to achieve different objectives. However, they do not necessarily address all of the open problems that we raise as below.

A. How to make a representation more robust to evading techniques?

The detector's resilience to evasion techniques is an issue that can determine the accuracy level of ML-based malware detection models. Our study will only consider evasion techniques as tricking an analyzing tool to make a malware appear benign while still executing its payload. For example, changing individual API calls (by replacing them with equivalent ones or adding dummy system calls) can easily alter system-call-sequence-based representations and transition graphs. This attack can evade the ML-based detector and yet preserve the functionalities of the malicious payloads [9], [13], [17]. On the other hand, by feeding a heterogeneous graph-based malware representation to an ML detector, we expect to mitigate such functionality-preserving evasion attacks. Furthermore, we expect such a representation to unveil evading techniques used in non-ML-based malware analysis, such as camouflage and mimicry attacks.

B. How to make a representation abstract enough to make it cross-platform?

A common issue with current malware detection is the scarcity of dataset on less-targeted mainstream OS such as macOS or Linux. This issue concerns all the aforementioned works that are platform-specific because they rely notably on system calls. To alleviate this problem, we propose to design a cross-platform representation that could be used to represent software behavior from multiple OSes. This would allow to learn to recognize a certain family of malware very common on a certain platform and then use this detector on a different platform where this family is much more rare. Reaching this goal relies on the semantic level of the representations, as abstracting the meaning of system calls is necessary for a cross-platform representation and the objects an OS handles are most often not specific to any particular platform. This will also be required to reach the goal of III-D.

C. What elements make the representation of an execution trace visually exploitable for a human expert?

A visual representation can help an expert quickly assess the behavior of a software and understand its features by organizing its information to facilitate the search. The representations presented in Section II could be improved in that regard. On the one hand, it is difficult to make logical connections between actions represented by a time series of

system calls. On the other hand, the system call transition graph has a manageable constant size but the expert cannot see the exact chaining of actions anymore, discarding any in-depth analysis. Representations that focus on system objects (files, network, etc.) can reveal how a software interacts with its environment. However, the expert may miss essential information because such graphs are generally too large. The most polished visual representation is the Virus Total's relation graph. However, it doesn't include the nature of interactions or the order of actions, so it is not suited for behavior analysis. The approach based on object monitoring is promising, but we expect the size of such representation to be the main challenge. Our line of research will investigate how to help the expert navigate such representation using, for example, recommendations, graph pruning, or graph collapsing.

D. How to ensure a representation reflects all the malicious actions of a malware infection event?

In an execution trace report, the information used for malware analysis might not be presented in an exploitable way i.e. the related information might not appear directly linked.

For instance, representations in system call sequences (with parameters or not) show that the order of actions is a relevant information. Indeed it is missing in the transition graph representations, as some of the related actions do not directly follow each other.

Furthermore, the representations that include the system call parameters show that what matters is not only the order of operations in the sequence, but also the order of operations on objects (files, IPs, locks, etc).

Moreover, many programs will use multiple threads and/or processes, and a sequence representation is designed to represent a single sequence of actions. Heterogeneous information graph representations address that, and besides, this allows to make links between these threads/processes, via the objects that they handle to interact. Thus, these objects represent the interactions a sequence of actions has with the system at a higher level.

Finally, Virus Total's representation shows that we cannot get rid of the actions. By only showing the related objects, and not their interactions, nor the order of interactions, this is not enough to explain what happened during an execution.

IV. Conclusion and Goal of the PhD

This article surveys what is done in terms of representations of software execution traces. It unveils four research questions that the literature deals with insufficiently (or sometimes not at all). The Ph.D. thesis of the first author will therefore focus on proposing a new representation that encompasses all suspicious actions while being readable for human experts. Furthermore, this representation based on higher semantics should be less prone to evasion techniques and general enough to be cross-platform.

References

[1] Cuckoo sandbox. https://cuckoosandbox.org/.

[2] Virustotal. https://www.virustotal.com/gui/home/upload.

[3] Alptekin, H., Yildizli, C., Savas, E., and Levi, A. TRAPDROID: Bare-Metal Android Malware Behavior Analysis Framework. In *2019 21st International Conference on Advanced Communication Technology (ICACT)* (PyeongChang Kwangwoon_Do, Korea (South), Feb. 2019), IEEE, pp. 664–671.

[4] Anderson, B., Quist, D., Neil, J., Storlie, C., and Lane, T. Graph-based malware detection using dynamic analysis. *J Comput Virol 7*, 4 (Nov. 2011), 247–258.

[5] AV-test. Av-test statistics, 2021. https://www.av-test.org/en/statistics/malware/.

[6] Bridges, R., Hernández Jiménez, J., Nichols, J., Goseva-Popstojanova, K., and Prowell, S. Towards Malware Detection via CPU Power Consumption: Data Collection Design and Analytics. In *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)* (Aug. 2018), pp. 1680–1684. ISSN: 2324-9013.

[7] Chen, Z.-G., Kang, H.-S., Yin, S.-N., and Kim, S.-R. Automatic Ransomware Detection and Analysis Based on Dynamic API Calls Flow Graph. In *Proceedings of the International Conference on Research in Adaptive and Convergent Systems* (Krakow Poland, Sept. 2017), ACM, pp. 196–201.

[8] Ding, Y., Xia, X., Chen, S., and Li, Y. A malware detection method based on family behavior graph. *Computers & Security 73* (Mar. 2018), 73–86.

[9] Fogla, P., Sharif, M. I., Perdisci, R., Kolesnikov, O. M., and Lee, W. Polymorphic blending attacks. In *USENIX security symposium* (2006), pp. 241–256.

[10] Grimmer, M., Röhling, M. M., Kricke, M., Franczyk, B., and Rahm, E. Intrusion Detection on System Call Graphs. *Sicherheit in vernetzten Systemen* (2018), 18.

[11] Kinable, J., and Kostakis, O. Malware Classification based on Call Graph Clustering. *Journal in Computer Virology 7*, 4 (2011), 233–245.

[12] Kolosnjaji, B., Zarras, A., Webster, G., and Eckert, C. Deep Learning for Classification of Malware System Call Sequences. In *AI 2016: Advances in Artificial Intelligence*, B. H. Kang and Q. Bai, Eds., vol. 9992. Springer International Publishing, Cham, 2016, pp. 137–149. Series Title: Lecture Notes in Computer Science.

[13] Kucuk, Y., and Yan, G. Deceiving Portable Executable Malware Classifiers into Targeted Misclassification with Practical Adversarial Samples. *Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy* (2020), 341–352.

[14] Park, Y., Reeves, D., Mulukutla, V., and Sundaravel, B. Fast malware classification by automated behavioral graph matching. In *Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research - CSIIRW '10* (Oak Ridge, Tennessee, 2010), ACM Press, p. 1.

[15] Piplai, A., Mittal, S., Abdelsalam, M., Gupta, M., Joshi, A., and Finin, T. Knowledge Enrichment by Fusing Representations for Malware Threat Intelligence and Behavior. In *2020 IEEE International Conference on Intelligence and Security Informatics (ISI)* (Nov. 2020), pp. 1–6.

[16] Sihwail, R., Omar, K., and Ariffin, K. A. Z. A survey on malware analysis techniques: Static, dynamic, hybrid and memory analysis, 2018.

[17] Stokes, J. W., Wang, D., Marinescu, M., Marino, M., and Bussone, B. Attack and Defense of Dynamic Analysis-Based, Adversarial Neural Malware Detection Models. In *MILCOM 2018 - 2018 IEEE Military Communications Conference (MILCOM)* (Los Angeles, CA, Oct. 2018), IEEE, pp. 1–8.

[18] Wang, S., Chen, Z., Yu, X., Li, D., Ni, J., Tang, L.-A., Gui, J., Li, Z., Chen, H., and Yu, P. S. Heterogeneous Graph Matching Networks for Unknown Malware Detection. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence* (Macao, China, Aug. 2019), International Joint Conferences on Artificial Intelligence Organization, pp. 3762–3770.

[19] Wunderlich, S., Ring, M., Landes, D., and Hotho, A. Comparison of System Call Representations for Intrusion Detection. *Springer, Cham 951* (2020), 14–24.

[20] Xiao, F., Lin, Z., Sun, Y., and Ma, Y. Malware Detection Based on Deep Learning of Behavior Graphs. *Mathematical Problems in Engineering 2019* (Feb. 2019), 1–10.