Journal of Information Security and Applications 37 (2017) 91–100

Contents lists available at ScienceDirect

Journal of Information Security and Applications

journal homepage: www.elsevier.com/locate/jisa

Classification of malware families based on runtime behaviors

Abdurrahman Pektaş*, Tankut Acarman

Galatasaray University, Computer Enginnering Department, Ciragan Cad No:36, 34349, Ortakoy, Istanbul, Turkey

ARTICLE INFO

Article history:

Keywords:
Behavior analysis
Dynamic analysis
Malware classification
Machine learning

ABSTRACT

Classification of malware samples plays a crucial role in building and maintaining security. Design of a malware classification system capable of supporting a large set of samples and adaptable to model changes at runtime is required to identify the high number of malware variants. In this paper, file system, network, registry activities observed during the execution traces and n-gram modeling over API-call sequences are used to represent behavior based features of a malware. We present a methodology to build the feature vector by using run-time behaviors by applying online machine learning algorithms for classification of malware samples in a distributed and scalable architecture. To validate the effectiveness and scalability, we evaluate our method on 17,900 recent malign codes such as viruses, trojans, backdoors, worms. Our experimental results show that the presented malware classification's training and testing accuracy is reached at 94% and 92.5%, respectively.

1. Introduction

Malicious software, or malware, is software used or created by an attacker to execute his/her bad intentions on a computer system without authorization and knowledge of its user. Basically, malicious attacks are targeted to steal safety-critical or liability-critical personal data or damage the compromised system. The recent developments in the field of computation system and proliferation of system such as smart phones, tablets, Internet of Things (IoT), cloud computing have led to an increased interest in malware development. The majority of new malware samples can be deployed as the variant of the previously known samples. The proliferation of the runtime packer and obfuscation techniques easily enables creation of behaviorally identical but statically different malware samples [1]. According to [2], more than 430 million new unique pieces of malware were detected in 2015 with an increase of 36% from the previous year, and a new zero-day vulnerability was discovered at each week on average with a doubled release frequency in comparison with the previous year.

Malware analysis can be grouped into two main categories based on whether or not the file under scrutiny is executed during the analysis: the associated methods can be referred as signature-based or behavior-based. Signature-based methods rely on unique raw byte patterns or regular expressions, known as signatures, created to match the malicious file. For instance, static features of a file are used to determine whether it is a benign or a malware. The main advantage of signature-based methods is their exhaustiveness since they trace all possible execution paths of a given file. Although these methods provide good detection rate on known samples, they are vulnerable to code obfuscation techniques such as run-time packing, metamorphism, and polymorphism that is generally used by malware authors to evade detection [3].

Unlike signature-based methods, behavior-based approaches require execution of a given sample in a sandboxed environment and run-time activities are monitored and logged. Dynamic analysis frameworks employ both virtualization and emulation environments to execute a malware and to extract its behaviors [4,5]. The behavior of an executable is extracted either by monitoring system changes made in the OS, or tracking API calls along with their parameters and returning values during execution. Although monitoring system changes is necessary to analyze behavior of a malware, this scheme does not involve monitoring some important behaviors (i.e., searching for specific file types or file name, enumeration of special registry keys, etc.,) adopted by advanced malware samples, for instance anti-VM technique to thwart analysis. Besides, some research efforts have focused on extracting behaviors based on the state changes between clean and dirty snapshots [6,7], where a clean (**dirty**) snapshot is a state of the machine before (**after**) execution of a malware sample.

Recently, for malware detection and determination whether being benign or malicious software, researchers have applied a fixed size n-gram and variable length n-gram that can be extracted from the binary content of the analysis file and opcodes obtained after dissembling [8,9]. For instance, a sequence of opcodes is used to

* Corresponding author.
E-mail addresses: apektas@yandex.com (A. Pektaş), tacarman@gsu.edu.tr (T. Acarman).

https://doi.org/10.1016/j.jisa.2017.10.005
2214-2126/

create a feature vector and three classifiers named as Ripper, C4.5 and IBK are used with ensemble learning algorithm to improve the accuracy in classification [10].

Meanwhile, function-call, control-flow, and data-flow graphs, which are more robust to code obfuscation than n-gram, are introduced for malware detection and classification [11]. In graph mining approaches, given software is simply presented as a graph. Then, this graph is compared with training graphs to identify the most similar one found in the dataset. Since graph matching is computationally expensive (an NP hard problem), graph comparing algorithms have been proposed to differentiate maliciousness from benign graphs.

Also, API calls reflect the aim of a program, and analyzing these calls can reveal the behavior of a program with less computational resource requirements. There are two methods to obtain the list of API calls: static analysis (e.g., IDA Pro-disassembly tool) or dynamic analysis (e.g., API hooking). Since a software can include multiple execution paths, dirty, and unused codes, extraction of API calls through static analysis by disassembler (e.g., with IDA Pro) is a challenging task. Moreover, disassemblers can be evaded by anti-disassembly methods. Last but not least, manual analysis of these calls can be a tedious task since a simple executable can make a considerably large amount of API calls. However, if a malware does not feature run-time protection, one can accurately obtain API calls through dynamic analysis.

In open literature, these methods are applied to detect and classify a malware by using different number of samples. The approach in [12] presents classification system based on a n-gram feature vector extracted from network level artifacts obtained via dynamic analysis. The evolution set of this work consists of 3 families and includes around 3000 samples. By using SVN, k-NN and decision tree, 80% accuracy in classification is achieved. In [13], the API calls and their arguments are used to model behavior. But evaluation is made with a limited amount of malware samples. A pre-defined set of API calls and a narrow feature space built to represent a software is used in [14] but crucial information about behavior is not extracted due to poor modeling of a malware. In [15], a behavioral fingerprint of a malware is composed of system state changes such as files written, processes created and rather than sequences or patterns of system calls. To measure similarity among the malware groups, a tree structure based on single-linkage clustering algorithm is presented. The method is tested by using real world malware samples (including samples that have not been detected yet, and therefore do not have a signature) and more successful classification results are obtained in comparison with anti-viruses using signature-based methods.

In [16], a classification method is introduced in order to determine whether a given malware sample is a variant of known malware family or a new malware strain. System call traces are captured and the behavior of malicious software is monitored by means of special representation called Malware Instruction Set (MIST), which is inspired from instruction sets used in CPU. In this representation, the behavior of a sample is characterized with a sequence of instructions. A behavior-based automated classification method, which is motivated by [16], is proposed in [17]. Dynamic analysis report gives the status change caused by the executable and events, this information is obtained from corresponding Win32 API calls and their certain parameters. Behavior unit strings are extracted as the features in order to distinguish malware families. To reduce the dimension of feature space, string similarity and information gain measure is used. A malware classification method using runtime actions and API calls of malware samples is presented in [18]. Supervised machine learning Random Forests is applied with 160 trees to classify 42,000 malware samples into 4 malware families. True positive rate is reached at 0.896 and false positive rate at 0.049 subject to the restricted number of families. In [19], malware samples are detected first and then classified as unknown or known malware by applying Random Forests classifier. Behavioral traces and API calls along with input parameters are used to build the feature vector. 31,295 malware samples belonging to 5 families and 837 benign samples are used for determining whether they are known or unknown, true positive rate and false positive rate is reached at 0.981and 0.099 subject to the 5 respective malware families. In [20], malware detection based on API call sequence analysis is presented. Malware samples are executed in a virtual environment and API call sequences are traced during runtime by using user-space hooking library called Detour [21]. Then, DNA sequence alignment technique is applied to remove meaningless codes inserted into malware samples. Finally, the common API call sequence patterns among malware are extracted by applying the longest common sub-sequences (LCS) algorithm. 2727 kinds of API into 26 groups are categorized in accordance with MSDN library. Classification accuracy is reached at 99% as the result of testing dataset consisting of 6910 malware and 34 benign samples. The main limitation of this method is that computing LCS and DNA sequence alignment is NP-hard problem, therefore computational complexity is high requiring more computational resources and time.

Throughout evaluation of a malware and application of online machine learning algorithms, a trade-off exists between the scalability of large-scale malware classification and computational complexity. For instance, when the feature space increases, data become sparse and the computation time of algorithms increases exponentially with the number of malware samples making the analysis inefficient. This problem is also known as the *curse of dimensionality*.

In this paper, we present a malware classification methodology while grouping samples based on their runtime behavior patterns by applying online machine learning. We capture implicit features of behavior in order to improve the accuracy of classifying malware. We perform an extensive assessment of our technique using standard classification evaluation metrics (e.g., accuracy, precision, recall, F1-score) and a large number of malware families, showing favorable evaluation results. Furthermore, we present the computational resource usages needed to deploy the presented classification methodology. A preliminary version of this study was presented in [22] and run-time behaviors were extracted to build the feature vector. Compared to [22], additional results about using API call sequences, resource usage with a more complete background are elaborated.

The rest of the paper is organized as follows: Section 2 describes the methodology for extracting behavior of the file under analysis along with the implementation details. Experiments and their results are discussed in Section 3. Conclusions and limitations are given in Section 4.

2. Methodology and implementation

The proposed framework consists of three major stages. The first stage consists of extracting the behavior of the sample file under scrutiny and observing its interactions with the OS resources. At this stage, the sample file is run in two sandboxed environment; VirMon [4] and Cuckoo [5]. During the second stage, we apply feature extraction to the analysis report. The label of each sample is determined by using Virustotal [23]. Then at the final stage, the malware dataset is partitioned into training and testing set. The training set is used to obtain a classification model and the testing set is used for evaluation purposes. An overview of our system including its main functionalities is presented in Fig. 1.

From the viewpoint of this study, the run-time behavior of a given file is modeled by fusing both API calls and changes made