COMPUTERS & SECURITY 000 (2018) 1–15

Available online at www.sciencedirect.com

ScienceDirect

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

Computers & Security

Malware identification using visualization images and deep learning

Sang Ni[a], Quan Qian[b,a,*], Rui Zhang[a]

[a] School of Computer Engineering & Science, Shanghai University, Shanghai 200444, China
[b] Shanghai Institute for Advanced Communication and Data Science, Shanghai University, Shanghai, China 200444

ARTICLE INFO

Article history:
Received 12 June 2017
Revised 27 March 2018
Accepted 7 April 2018
Available online xxx

Keywords:
Network security
Malware
Visual analysis
Deep learning

ABSTRACT

Currently, malware is one of the most serious threats to Internet security. In this paper we propose a malware classification algorithm that uses static features called *MCSC* (Malware Classification using SimHash and CNN) which converts the disassembled malware codes into gray images based on SimHash and then identifies their families by convolutional neural network. During this process, some methods such as multi-hash, major block selection and bilinear interpolation are used to improve the performance. Experimental results show that *MCSC* is very effective for malware family classification, even for those unevenly distributed samples. The classification accuracy can be 99.260% at best and 98.862% at average on a malware dataset of 10,805 samples which is higher than other compared algorithms. Moreover, for *MCSC*, on average, it just takes 1.41 s to recognize a new sample, which can meet the requirements in most of the practical applications.

1. Introduction

Malware is one of today's major Internet security threats. An Internet security threat report from Symantec (2016) shows that more than 430 million unique pieces of malware were discovered in 2015, an increase of 36% from the year before. The amount of new malware has been continuously growing, and its threats are increasing rapidly.

Malware, which is slipped into a victim's computer by hackers (attackers) through security vulnerabilities of the operating system or application software, can influence normal operation, collect sensitive information and steal superuser privileges in order to perform malicious actions. Generally, mainstream malware includes malicious scripts, vulnerability exploits, back doors, worms, trojans, spywares, rootkits, etc., and combinations or variations of the above types as well.

Traditionally, most malware detection systems are based on feature vectors, which contain essential characteristics of malware. Mainstream malware feature extraction can be divided into two categories, static and dynamic. In static analysis, malicious software is analyzed without executing it (Gandotra et al., 2014). The detection patterns used in static analysis include string signature, byte-sequence n-grams, syntactic library call, control flow graph and opcode (operational code) frequency distribution. For static analysis, the executable has to be unpacked and decrypted in advance.However, in dynamic analysis, malicious software is analyzed while being executed in a controlled environment (e.g., virtual machine, simulator, emulator, or sandbox). Before executing the malware samples, appropriate monitoring tools like Process Monitor or Capture BAT are installed and activated. While static analysis is usually vulnerable to code obfuscation, dynamic analysis is time consuming and computationally intensive.

The main contributions of this paper are: (1) Proposing a novel *MCSC* algorithm which combines opcode sequence and *LSH* to extract malware features. During feature extraction,

* Corresponding author.
E-mail addresses: qqian@shu.edu.cn, qqian@i.shu.edu.cn (Q. Qian).
https://doi.org/10.1016/j.cose.2018.04.005

major block selection is used to extract the main part of malware. The preprocessing can reduce the influence from obfuscation in other parts and reduce calculation in following steps. (2) Using visualization techniques to transform the malware's non-intuitive features into fingerprint images. Compared to the original opcode sequence, a Simhash of the same size can easily be converted into an image. (3) Using a *CNN* to train the malware fingerprint images and get excellent identification results. (4) Improving *MCSC* further using multi-hash, major block selection and bilinear interpolation.

This paper is organized as follows. Malware-related studies are reviewed in Section 2. In Section 3, the *MCSC* algorithm is proposed and discussed in more detail. The experimental results are presented in Section 4, and Section 5 summarizes the whole paper.

2. Related work

Malware detection methods are typically divided into two categories: static analysis and dynamic analysis. In static analysis, the malware binary file is disassembled or decompiled without executing it. Thus, static analysis reveals the malware's behavior while preventing the operating system from malicious damages. However, in most cases static analysis is not a trivial task since attackers use code obfuscation techniques such as binary packers, encryption or self-modifying techniques to evade static analysis. Furthermore, static analysis does not allow a high degree of automation during analysis. In dynamic analysis, the behavior of malware is analyzed during execution in a debugger. Currently, sandbox-based dynamic analysis is one of the most promising techniques. A sandbox executes a malware sample in a controlled environment that can monitor and record information of system calls and behaviors dynamically. The main limitations of dynamic analysis, especially sandbox-based solutions, are the extensive and detailed reports requiring human analysis and interpretation. Furthermore, in contrast to static analysis, dynamic analysis can be automated to a high degree, and it also has high computation complexity.

2.1. Static analysis

In static analysis, Schultz et al. (2000) were the first to introduce the concept of data mining for detecting malware. They used three different static features for malware classification: Portable Executable (PE), strings and byte sequences. A rule induction algorithm called Ripper (Cohen, 1995) was applied to find patterns in the DLL. A Naive Bayes algorithm was used to find patterns in the string data, and *n*-grams of byte sequences were used as input data for the Multinomial Naive Bayes algorithm. Tian et al. (2008) used function length frequency to classify Trojans. Zolkipli and Jantan (2011) used variable length instruction sequences along with machine learning for worm detection. They tested their method on a dataset with 1444 worms and 1330 benign files. Kong and Yan (2013) presented a framework for automated malware classification based on structural information (function call graph) of malware. They used an ensemble of classifiers that learn from pairwise malware distances to classify malware into their respective families. Santos et al. (2013a) proposed a method for representing malware that relied on opcode sequences in order to construct a vector representation of the executables. Shankarapani et al. (2011) proposed two general malware detection methods: Static Analyzer for Vicious Executables (SAVE) and Malware Examiner using Disassembled Code (MEDiC). Their experimental results indicate that both of their proposed techniques can provide better detection performance against obfuscated malware. Gu et al. (2015) proposed a malicious document detection method based on wavelet transform and designed a malicious detection system based on this method. Li and Li (2015) proposed a 3-layer description of API-calls and similarity comparison method based on static code structure to determine what code family a malicious Android APK belongs to. This method can be used for evaluation and classification of unknown APKs.

2.2. Dynamic analysis

Dynamic analysis generally includes tainting, behavior-based methods, and API call monitoring (Han et al., 2014). Bayer et al. (2009) proposed a system that clusters large sets of malicious binaries based on their behavior effectively and automatically. Zolkipli and Jantan (2011) presented an approach for malware behavior analysis. They used *HoneyClients* and *Amun* as security tools to collect malware. Behavior of these malware were then identified by executing each sample on 2 virtual platforms, *CWSandbox* (Anderson et al., 2011) and *Anubis*. Anderson et al. (2011) presented a malware detection algorithm based on the analysis of graphs constructed from dynamically collected instruction traces. Imran et al. (2015) proposed a malware classification scheme based on Hidden Markov Models using system calls as observed symbols. Fujino et al. (2015) proposed "API call topics", a kind of API call behavior for a malware family, to identify similar malware samples. They applied an unsupervised non-negative matrix factorization (NMF) clustering analysis to extract API call topics from a large corpus of API calls, which can detect similar malware samples. Lim et al. (2015) proposed a malware classification method based on network flow activity. They used clustering of flow features and a sequence alignment algorithm (generally used in bio-informatics to compare two or more character sequences to obtain their similarities) to analyze the malware traffic flow behavior. The main limitations of dynamic analysis, especially the sandbox-based solutions, are that some malware can detect and change behavior when running in virtual environments. Therefore, dynamic analysis might not always uncover malicious behavior. Additionally, it is difficult to ensure execution path coverage using dynamic analysis.

2.3. Visualization analysis

Recently, several visualization techniques have been proposed for malware analysis. Yoo (2004) used Self-Organizing Map to visualize and detect viruses. Quist and Liebrock (2009) presented a VERA framework to visually represent the overall flow of a program, which depends on Ether hypervisor to covertly monitor the program execution based on dynamic analysis. Trinius et al. (2009) visualized the behavior

Please cite this article as: Sang Ni et al., Malware identification using visualization images and deep learning, Computers & Security (2018), https://doi.org/10.1016/j.cose.2018.04.005