Neurocomputing 151 (2015) 905–912

Contents lists available at ScienceDirect

Neurocomputing

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

Android based malware detection using a multifeature collaborative decision fusion approach

Shina Sheen *, R. Anitha, V. Natarajan

Department of Applied Mathematics and Computational Sciences, PSG College of Technology, Coimbatore, India

ARTICLE INFO

Article history:
Received 15 February 2014
Received in revised form
9 September 2014
Accepted 3 October 2014
Communicated by T. Heskes
Available online 18 October 2014

Keywords:
Android
Malware
Multifeature
Collaborative decision fusion

ABSTRACT

Smart mobile device usage has expanded at a very high rate all over the world. Since the mobile devices nowadays are used for a wide variety of application areas like personal communication, data storage and entertainment, security threats emerge, comparable to those which a conventional PC is exposed to. Mobile malware has been growing in scale and complexity as smartphone usage continues to rise. Android has surpassed other mobile platforms as the most popular whilst also witnessing a dramatic increase in malware targeting the platform. In this work, we have considered Android based malware for analysis and a scalable detection mechanism is designed using multifeature collaborative decision fusion (MCDF). The different features of a malicious file like the permission based features and the API call based features are considered in order to provide a better detection by training an ensemble of classifiers and combining their decisions using collaborative approach based on probability theory. The performance of the proposed model is evaluated on a collection of Android based malware comprising of different malware families and the results show that our approach give a better performance than state-of-the-art ensemble schemes available.

1. Introduction

Mobile technology has expanded dramatically around the world. Nowadays smart mobile devices are ubiquitous, they are used for personal mobile communication, data storage, multimedia and entertainment. These devices integrate the functionality of PDAs and mobile phones and make different types of network services available to mobile applications. According to the Cisco Visual Networking Index [1], global mobile data traffic will increase 18-fold between 2011 and 2016. By the end of that time period, it is projected that there will be 8 billion mobile devices in use around the world. Android and Apple iOS combined account for the significant majority of the global smartphone installed base in 2012. Smartphones are often used to read enterprise e-mail and documents, find local businesses, get deals on products, buy them and click on mobile ads. This may lead to a large number of devices being vulnerable to varied types of attacks. As most of the critical applications like banking, E-Commerce, etc., are moving to mobile environment, a malicious attack not only puts in jeopardy personal privacy, but also threatens a financial disaster. Android has attracted the most consideration of malicious elements due to its growing popularity and openness as compared to iOS and Windows. Juniper Networks in their third annual Mobile threats report [2] has stated that malware aimed specifically at Android devices have increased at an alarming rate since 2010. Cisco's 2014 Annual Security Report states that 99% of mobile malware software target Android platform [3].

The two approaches of malware analysis, static and dynamic, widely used in the literature has been used in Android based malware detection also. Although dynamic analysis of Android malware can provide a comprehensive view, it is still subjected to high cost in environment deployment and manual efforts in investigation. In this study, we propose a static feature-based mechanism for detecting Android malware. It is seen that multiple types of features like API calls and permissions requested by an application can be extracted from an Android apk (Android application package) file. Learning from these representations separately can lead to better gains than considering them as a single dataset. In this work, a multifeature approach by extracting various features from Android apk files is considered. Our method is particularly appealing for the following three reasons:

- First, we are able to combine different feature sets of an executable file in a way which allows the learning algorithm to take advantage of all the features simultaneously.
- Second, our method is scalable in the sense that future popular data sources could be easily added to the model without complicating the final result.

* Corresponding author.
E-mail addresses: shina_np12@yahoo.com (S. Sheen), anitha_nadarajan@mail.psgtech.ac.in (R. Anitha), natarajan.v.in@ieee.com (V. Natarajan).

http://dx.doi.org/10.1016/j.neucom.2014.10.004
0925-2312/

- Finally, the method is based on a generalized collaborative approach where there are n different classifiers and m different feature sets. The classifiers collaborate with each other to reach a final decision rather than depending on a single classifier.

The main contributions in this work are

- Analysis of a large collection of malicious and benign files to find the most discriminating features for malware detection.
- Consideration of multiple sets of features rather than depending only on a single data source.
- Design and implementation of an ensemble of classifiers exploiting the multiple sets of features extracted from Android files using a collaborative method for decision fusion.

The remainder of this paper is organized as follows. Section 2 discusses the related work and Section 3 discusses in detail about the various features extracted from executable files. Section 4 describes the proposed method to classify a given file based on Multifeature Collaborative Decision Fusion (MCDF). The experimental results and analysis are presented in Section 5 followed by the concluding remarks.

2. Related work

With the proliferation of mobile devices in the market there has been an active research in the field of Android based malware detection in the recent past. The APK file is the file format used to distribute and install software (usually games or applications) on the Android operating system. Every application must have an AndroidManifest.xml file in its root directory. The manifest presents essential information about the application to the Android system, before it can run any of the application's code. It declares which permissions the application must have in order to access protected parts of the API and interact with other applications. In order to protect Android users, applications access to resources is restricted with permissions. An application must obtain permissions in order to use sensitive resources like the camera, microphone, or call log. Felt et al. [4] developed a tool that generates the maximum set of permissions required for an application and compares them to the set of permissions actually requested. The Kirin project [5] provides lightweight certification of applications at the time of installation by looking at the configuration metadata such as requested permissions, which accompanies Android applications. From this metadata, Kirin infers potential functionality and compares it against a ruleset of potentially dangerous properties. If any rule fails, the application is not installed.

In [6] a machine learning based system for the detection of malware on Android devices is presented. Permissions and control graph features are extracted and trained on a one-class support vector machine in an offline manner. But real time detection with control flow graphs is a challenging task. Sanz et al. [7,8] analyzed the user permissions and other user features and applied various machine learning algorithms to attain an accuracy of 86%. Zhou et al. [9] identified Android malware based on the similarities of the requested permissions and also on the behavioral aspects like the installation method, nature of carried payload to known malware families.

Another area of research is in the analysis of system calls. Schmidt et al. [10] extracted library and system function calls from Android executables and compared them to malware executables to classify apps. Crowdroid [11] collected system call traces of running apps on different Android devices. They applied clustering algorithms to detect malware. In DroidAPIminer [12] frequency analysis of API calls within benign and malware apps are made and evaluated on different classifiers using the generated feature set. Andromaly [13] is designed to continuously monitor various system metrics to detect suspicious activities through applying supervised anomaly detection techniques.

Bartel et al. [14] analyzed various applications and concluded that many applications declare permissions but are not actually used. So just by mining the manifest file for permissions alone may not give an accurate result. In an Android framework, once an application is installed, a set of APIs are called during runtime. Each API call is associated with a particular permission. When an API call is made, the Android OS checks whether or not its associated permission has been approved by the user. Only if a match results, it will proceed to execute the certain API call. So the API calls provide the information whether a permission is actually used or not.

The work on ensemble based learning, where combining the decisions of a collection of classifiers can be an effective strategy [15], is an interesting area that has got a lot of attention. Recently Wozniak et al. [16] have given a comprehensive study of the advantages of multiple classifier systems and its applications in varied fields like remote sensing, computer security, financial risk assessment, fraud detection, medical diagnosis and recommender systems. According to Axelsson [17], there are many different types of intrusions in reality, and different detectors are needed to detect them. If one method or technique fails to detect an attack, then another should detect it. Lee [18] proposed that combining multiple models allows for a more easily adaptable and extendible framework. Fan [19] expanded on this idea, noting that an ensemble approach allows one to quickly add new classifiers to detect previously unknown activity. His research in this area suggested that this can be done without any loss, and possibly a gain, in classification performance. An ensemble of one class classification is proposed by [20–22] where the number of training instances of the target class are very less. Jackowski et al. [23] proposed an effective training procedure consisting of two phases. The first phase detects the classifier competencies and adjusts the respective fusion parameters. The second phase boosts classification accuracy by elevating the degree of local specialization. Robert et al. [24] proposed a collaborative decision fusion using n views and n agents for fish classification. In their work one view is given to one agent and then collaborative fusion is done.

We have extracted feature sets mainly in two categories namely the permissions required for an application and the API calls used within an application. The different types of data sources mentioned above try to capture the relation between the permissions declared in the manifest file and the permissions actually used. Considering both these features may produce a more accurate result. Our work differs from the above methods in the sense that we consider two discriminating features together and train them on separate classifiers which may have different generalization errors. Further a collaborative decision is made based on the joint probabilities of classification by the various classifiers.

3. Feature extraction

This section describes the analysis of Android apk files and the various features extracted. Using static analysis we have extracted features like permissions and API calls.

3.1. Permissions

Android OS model requires applications to highlight what features of OS or resources they are going to use. By default android applications do not have any permission to intervene the device data or components. This can be provided either in ⟨application⟩ tag or ⟨uses – permission⟩ in the manifest file, a sample of which is shown in Fig. 1.