Future Generation Computer Systems 36 (2014) 122–132

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Mining permission patterns for contrasting clean and malicious android applications

Veelasha Moonsamy, Jia Rong*, Shaowu Liu

School of Information Technology, Deakin University, 221 Burwood Highway, Vic 3125, Australia

HIGHLIGHTS

- A pattern mining algorithm is proposed to identify contrast permission patterns.
- We collected a new dataset with 1227 clean Android applications.
- We considered both required and used permission.
- Biclustering method has been employed to provide visualization.
- The permission patterns show big contrasts between clean apps and malware.

ARTICLE INFO

Article history:
Received 16 March 2013
Received in revised form
1 August 2013
Accepted 5 September 2013
Available online 18 September 2013

Keywords:
Android permission
Data mining
Biclustering
Contrast mining
Permission pattern

ABSTRACT

An *Android* application uses a permission system to regulate the access to system resources and users' privacy-relevant information. Existing works have demonstrated several techniques to study the required permissions declared by the developers, but little attention has been paid towards used permissions. Besides, no specific permission combination is identified to be effective for malware detection. To fill these gaps, we have proposed a novel pattern mining algorithm to identify a set of contrast permission patterns that aim to detect the difference between clean and malicious applications. A benchmark malware dataset and a dataset of 1227 clean applications has been collected by us to evaluate the performance of the proposed algorithm. Valuable findings are obtained by analyzing the returned contrast permission patterns.

1. Introduction

Smartphone is used to describe a mobile device equipped with enhanced computing capability and connectivity [1], such as *Nexus* by *Google* [2], *iPhone* by *Apple* [3], *Blackberry* by *RIM* [4] and *Windows Phone* by *Microsoft* [5]. In the past few years, the global telephony industry has witnessed an upsurge in the sales of smartphones. A smartphone is usually sold with an in-built mobile operating system (OS) together with a number of pre-installed "applications" packaged by the device manufacturer. An *application*, the software running on smartphones, enhances the smartphone's functionality and supports the interaction with end users to accomplish their tasks. *Calendar*, *address book*, *alarm clock*, *media player* and *web browser* are the common applications provided by the device manufacturers, but one important application exists on every smartphone—the "*application store*", which allows end users to access *online application markets* to browse and download additional applications of their choice.

Every device manufacturer hosts an application market for its own OS platform, such as *Apple*'s *App Store* [6], *Blackberry*'s *App World* [7] and *Google Play* [8]. However, far before the first *official application markets* were introduced in 2008 by *Apple*, smartphone application distribution was highly dependent on *third-party* sources, where individual application developers were free to upload their products. Due to a huge number of low-price applications being available, there is still a large group of end users who prefer visiting *third-party application markets*, but not all the applications from markets are "*safe*". The software that is specially designed to harm a device, its OS or other software is called "*Malware*", which stands for *mal*icious soft*ware* [9]. The increasing sales of smartphones has pushed the rapid growth of mobile malware.

As pointed out by Zhou and Jiang [10], malware or malicious applications might cause a series of user unexpected operations, for example, stealing user's personal information, making calls or sending an SMS without the user's knowledge. Such malicious behaviors not only cost users extra data usage, but also potentially bring privacy issues. Furthermore, the users may not be aware of

* Corresponding author. Tel.: +61 411645497.
E-mail addresses: v.moonsamy@research.deakin.edu.au (V. Moonsamy), jiarong@acm.org, jiarong@tulip.org.au (J. Rong), swliu@deakin.edu.au (S. Liu).

0167-739X/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.future.2013.09.014

running malware on their smartphones because in many cases the malware are downloaded and/or installed without authorization. Accordingly, an efficient and effective malware detection technique is highly demanded to protect smartphone users from the potential prevalence.

To effectively detect malware from millions of applications available on official and third-party markets, many efforts have contributed to studying the nature of smartphone platforms and their applications in the past decade. As the most popular mobile platform, *Google*'s *Android* overtook others to be the top mobile malware platform. The *Android* platform employs the permission system to restrict applications' privileges to secure the users' privacy-relevant resources [11]. An application needs to get a user's approval for the requested permissions to access the privacy-relevant resources. Thus, the permission system was designed to protect users from applications with invasive behaviors, but its effectiveness highly depends on the user's comprehension of permission approval. We refer to the permissions that are requested during application installation as *required permissions*. Unfortunately, not all the users read or understand the warnings of required permissions shown during installation. To improve this situation, many researchers have tried to interpret *Android* permissions and their combinations [12–15]. Frank et al. [11] proposed a probability model to identify the common required permission patterns for all *Android* applications. Zhou and Jiang [10] listed the top required permissions for both clean and malicious applications, but only individual permissions were considered by frequency counting. A problem is still remaining of whether the patterns in a permission combination can provide better performance for malware detection. Furthermore, in the existing literature, only the required permissions are considered in permission pattern mining, no work has incorporated the *used permissions* that are extracted from static analysis by the *Andrubis* system (http://anubis.iseclab.org) [16]. Therefore, we are the first group to explore both the required and used permissions. Accordingly, our aim is *to propose an efficient pattern mining method to identify a set of contrast permission patterns that effectively distinguish malware from the clean applications*.

By using a pattern mining technique to identify the desired permission patterns, we need two datasets: one has only clean *Android* applications and the other contains all malicious ones. In 2012, Zhou and Jiang [10] published the first benchmark dataset of malicious applications in 49 malware families, which was collected from third-party markets between August 2010 and October 2011. This is an ideal malware dataset for our experiments. On the other hand, due to the lack of a dataset of clean applications published at the same time period as Zhou and Jiang's, we collected our own clean dataset. The clean applications were collected from two popular third-party *Android* applications markets: *SlideME* (http://slideme.org) and *Pandaapp* (http://android.pandaapp.com). We sorted the collected applications based on the times of their download and the ratings given by the users, and only the top ones were picked. Each application was scanned by forty-three antivirus engines on *VirusTotal* (https://www.virustotal.com) [17], and only the ones that passed all virus tests were considered as "clean" and kept to form the clean dataset. These clean applications do not impede on the smooth execution of the OS. Like Zhou and Jiang, we represent applications in the collected clean dataset using a vector of 130 binary values, each of which is associated with one of the 130 official *Android* permissions. A value 1 is assigned to a permission only if it is required or used by an application, otherwise, 0 is given instead.

The novelty and contributions of this work can be summarized as follows:

- We collected a new dataset that contains 1227 clean applications that were uploaded to third-party markets from August 2010 to October 2011.
- Beyond the current studies that focused on *required permissions* only, we also considered the *used permissions*.
- We utilized a hierarchical *Biclustering* method to initially analyze both clean and malware datasets. The obtained resulting figures provided a straightforward preview of the data distribution, from which we built up our model of mining a set of permissions rather than using individual permissions as the patterns.
- We proposed a contrast permission pattern mining algorithm to identify the interesting permission sets that can be used to distinguish applications from malicious to clean.
- Our demonstration of the proposed *Contrast Permission Pattern Mining* proved that both required and used permissions should be considered in late malware detection tasks.

The rest of the paper is organized as follows: Section 2 briefly reviews the concepts of the *Android* platform, its applications, the permission system and the current research work in malware detection. In Section 3, we present our initial analysis on the collected datasets using a statistical method and *biclustering* followed by the proposed contrast pattern mining algorithm. The experiments and the obtained results are then reported in Section 4 followed by a further discussion on findings. Finally, Section 5 concludes the entire paper together with our future work.

2. Background and related work

2.1. Android

Android is a Linux-based OS which was designed and developed by the *Open Handset Alliance* in 2007 [18]. The *Android* platform is made up of multiple layers consisting of the OS, the Java libraries and the basic built-in applications [19]. Additional applications can be downloaded and installed from either official or third-party markets.

Google provides the application developer community with a *Software Development Kit* (SDK) [20] to build *Android* applications and it includes a collection of Java libraries and classes, sample applications and developer documentations. The SDK can be used as a plug-in for Eclipse IDE [21] and therefore allows developers to code their applications in a rich Java environment. One particularly useful feature of the SDK is the *Android emulator* which allows developers to test their applications in virtual devices on various versions of *Android* OS.

An *Android* application includes two folders and one file: (i) Class, (ii) Resources and (iii) AndroidManifest.xml. The Class folder contains the application's source code in Java; the Resources folder stores the multimedia files; and the `AndroidManifest.xml` file lists the required permissions that are declared by the developer. After the Java source code is done, it is then compiled and converted into Dalvik byte code [22] and bundled with the Resources folder and `AndroidManifest.xml` file to generate the *Android Application Package* (APK). Finally, before the APK can be installed on a device or emulator, the develop has to generate a key and sign the application.

Android developers can upload their applications to either the official market, *Google Play* [23], or any third-party market. To secure the privacy-relevant resources for its users, *Google* provides automatic antivirus scanning [24]. The applications will be rejected from *Google Play* if any malicious content is detected. From 2012, *Google* has extended its antivirus service on the new *Android* 4.2 OS, which is claimed to be able to scan applications before they are installed on the device [25].

2.1.1. Android permission system

Google applies the permission system as a measure to restrict access to privileged system resources. Application developers have to explicitly mention the permissions that need user's approval