CHEMICAL ENGINEERING TRANSACTIONS
VOL. 51, 2016
A publication of AIDIC
The Italian Association of Chemical Engineering
Online at www.aidic.it/cet
Guest Editors: Tichun Wang, Hongyang Zhang, Lei Tian

ISBN 978-88-95608-43-3; **ISSN** 2283-9216
DOI: 10.3303/CET1651116

An Android Malware Detection System Based on Cloud Computing

Shujuan Cui*, Gengxin Sun, Sheng Bin, Xicheng Zhou

Software Technical College of Qingdao University, Qingdao, China
qdu_csj@163.com

With the sharp increase in the number, smartphones are now ubiquitous. However, the security requirements of these new systems and the applications which they support are still being understood. Android platform, as a market leader, makes the need for malware analyses on the platform become an urgent issue. In this paper, a data mining approach through dynamic analysis of application behavior for detecting malware in the Android platform is proposed. We established a framework for collection of traces from an unlimited number of real users based on cloud computing, and behavior-related dataset of each application will be created. Then the partitional clustering algorithm is used to cluster each dataset. Through experimental results, the approach is shown to be an effective means of detecting the malware.

1. Introduction

Malware has threatened PCs for a long time, the increasing adoption of smartphones comes with the growing prevalence of mobile malware. As the most popular platform of smartphones, Android system has become the top malware platform.

Google's Android Market is the official online mechanism for delivering software to an Android based smartphones. Android application developers can upload their applications without any check of their trustworthiness. The applications are self-signed by developers themselves, without the intervention of any certification authority. There are also many unofficial repositories, where developers can upload applications more freely. The openness has allowed attackers to upload malware to the Market and to spread malware to user's smartphones very easily.

In this paper we propose a new approach to analyze the behavior of Android applications and distinguish between applications which have the same name and version but behave differently. The aim is to detect anomalously behaving applications, and then detecting malware.

The main goals and contributions of this paper are the use of a cloud computing platform to obtain the traces of applications' behavior, which would help researchers to collect different samples of application execution traces. These traces can then be used for leading to clear differentiation between the benign applications from malware.

This work is organized as follows. Section 2 describes related work. In Section 3 we explain structure of Android platform and four types of basic components. In Section 4 the behavior-based malware detection system framework is explained, especially detailing the process of building a cloud computing platform to collect and partitional clustering algorithm for detecting malware. In Section 5, we present the experimental results of the malware detection partitional clustering algorithm performed with a set of real malware applications. In Section 6 we conclude this paper.

2. Related work

Given the rampant growth of Android malware, there is a pressing need to effectively mitigate or defend against them. There are two approaches proposed for the analysis and detection of malware: static analysis (Asaf et al., 2009) and dynamic analysis (Gandotra et al., 2014). Static analysis is mostly based on source

Please cite this article as: Cui S.J., Sun G.X., Bin S., Zhou X.C., 2016, An android malware detection system based on cloud computing, Chemical Engineering Transactions, 51, 691-696 DOI:10.3303/CET1651116

code or binaries inspection looking at suspicious patterns. Although some approaches have been successful, the malware attackers have developed various effective techniques against static analysis.
On the other hand, dynamic analysis or behavior-based detection involves running the sample in a controlled environment in order to analyze its execution traces. Egele (Egele et al., 2012) provides a complete overview of automated dynamic malware analysis techniques. David Dagon (Dagon et al., 2004) alerted the community predicting the feasibility of malware in smartphones.
Due to the lack of smartphone malware patterns, most of anomaly detection techniques used the battery power consumption as the main feature of malware detection. These techniques were based on checking and monitoring smartphones power consumption and comparing them with the normal power consumption pattern to detect anomalies.
Because of resource limitations of smartphone, researchers proposed collaborative analysis techniques, where the analysis is made by a network of devices. Both static and dynamic analysis have been proposed using these techniques (Rubanov et al., 2014; Damopoulos et al., 2014).
Regarding Android Operating System, some authors provide overviews of its security model (Shabtai et al., 2010). One of the most important security measures of Android devices is the permission-based security model. Each application specifies which resources of the smartphone needs to be used, and the user grants or denies it's installation regarding the permissions needed. Jian (Jian et al., 2015) proposed a solution based on monitoring events occurring on Linux-kernel level. They reviewed Linux for enhancing security and extracting features such as system calls, from the Linux kernel. These features were then used to create a normal model for the smartphone behavior.
Both works use knowledge-based analysis while our system is behavior based. Our experimental results show that our system was capable of detecting every malware execution in self-written malware, giving a 100% of detection rate for this particular malware. We also provide results for the analysis and detection of real malware that can be found in the wild

3. Android overview

Android is a smartphone platform developed by the Open Handset Alliance. The platform quickly became popular among the developer community for its open source nature. Android provides a sophisticated message passing mechanism, in which Intents are used to link applications. An Intent is a message that declares a recipient and optionally includes data. It can be thought of as a self-contained object that specifies a remote procedure to invoke and includes the associated arguments. Applications use Intents for both inter-application communication and intra-application communication. Additionally, the Linux operating system sends Intents to applications as event notifications.

3.1 Basic components of Android

There are four types of components used to construct applications in Android, and each type has a specific purpose.
Activities which interface with the user via the touchscreen and keypad provide user interfaces. Only one Activity is active at a time, and processing is suspended for all other activities, regardless of the application. They can be started with Intents, and return data to their invoking components upon completion. All visible portions of applications are Activities.
Services run in the background and do not interact directly with the user. It provides background processing for use when an application's Activities leave focus. Other components can bind to a Service, which lets the binder invoke methods that are declared in the target Service's interface. Intents can be used to start and bind to Services.
Broadcast Receivers receive Intents sent to multiple applications. Receivers are triggered by the receipt of an appropriate Intent and then run in the background to handle the event. Receivers are typically short-lived, they often relay messages to Activities or Services. There are three types of broadcast Intents: normal, sticky, and ordered. Normal broadcasts are sent to all registered Receivers at once, and then they disappear. Ordered broadcasts are delivered to one Receiver at a time; also, any Receiver in the delivery chain of an ordered broadcast can stop its propagation. Broadcast Receivers have the ability to set their priority level for receiving ordered broadcasts. Sticky broadcasts remain accessible after they have been delivered and are re-broadcast to future Receivers.
Content Providers are databases addressable by their application-defined URIs. They are used for both persistent internal data storage and as a mechanism for sharing information between applications.
Intents can be sent between three of the four components: Activities, Services, and Broadcast Receivers. Intents can be used to start Activities; start, stop, and bind Services; and broadcast information to Broadcast Receivers.
The relations between these components are shown as Figure 1.

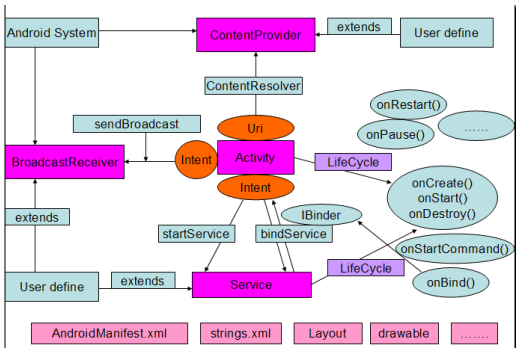

Figure 1: The relations between these components

3.2 Linux kernel of Android

Android's kernel is based on one of the Linux kernel's long-term support branches. The specific kernel version depends on the actual Android device and its hardware platform. Since April 2014, Android devices mainly use versions 3.4 or 3.10 of the Linux kernel.

In Linux, a system call is how a program requests a service from the operating system's kernel. Linux kernel 3.4 has more than 250 system calls and each one is identified by a unique number that is written in the kernel's system call table. System calls provide useful functions to application programs like network, file, or process related operations. As shown in Figure 2, when an application from user space makes a request to the Operating System, the petition goes through glibc library, System Call Interface, Kernel and finally to Hardware. glibc library interprets the petition and CPU switches to kernel mode to execute the appropriate kernel function looking into system call table. The kernel will be responsible for understanding the petition and making the request to the hardware platform. Afterwards the user gets the information requested by the application in the user space in an inverse process. Functions like getpid(), open(), read() and socket() are some of the functions that glibc can provide applications to invoke a system call.

Linux kernel is executed in the lowest layer of Android architecture. This means that all requests made from upper layers pass through the kernel using system call interface before they're executed in hardware. Capturing and analyzing the system calls that pass through system call interface, will provide accurate information about the behavior of the application.

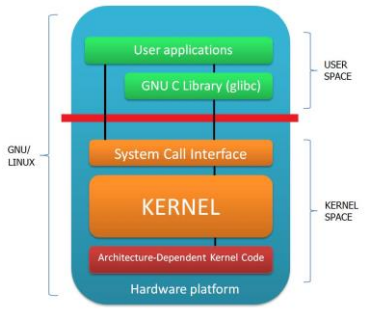

Figure 2: Hierarchy of Linux kernel and user space

4. Behavior-based malware detection system based on cloud computing

Security mechanisms used in computers are not feasible for applying on smartphones due to the excessive resource consumption and battery depletion. Hence, we decided to perform the whole analysis process on a dedicated cloud computing platform. This platform will be used exclusively to collect information and detect malicious and suspicious applications in the Android platform.

The whole system is composed of several components which provide enough resources to detect malware on the Android platform. First, we have developed a lightweight Application, which can be downloaded and installed from Google's official Market and unofficial repositories. The application is in charge of monitoring Linux Kernel system calls and sending them pre-processed to a centralized server. By using the application, users will help with sending non-personal, but behavior-related data of each application they use. The whole Behavior-based malware detection system is shown in Figure 3.

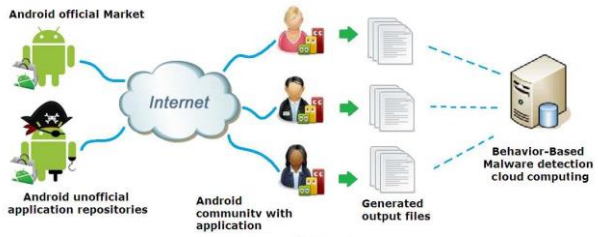

Figure 3: Framework of behavior-based malware detection system

The cloud computing platform will be in charge of parsing data, and creating a system call vector per each interaction of the users within their applications. Thus, a dataset of behavior data will be created for every application used. The more users using our application, the more complete and accurate will be our system. Each dataset will be clustered by using a partitional clustering algorithm. By using our system, we can differentiate between benign applications that demonstrate very similar system call patterns, and malicious applications that, even if having the same name and identifier, have a different behavior in terms of distance between example vectors.

Partitional clustering is simply a division of the set of data objects into non-overlapping clusters such that each data object is in exactly one cluster. Each cluster may be represented by a centroid or a cluster representative. Because we know that an Android application will be benign or malicious, so k-means algorithm (Jiaorao et al., 2015) is chosen due to its simplicity, efficiency and a known number of two clusters as an input parameter.

An example of an Android application behavior system call feature vector is shown as follows:

0 , 0 , 0 , 25 , 47 , 4 , 34 , 0 , 0 , 0 , 0 , 0 , 0 , 12 , 0 , 0 , 0 , 0 , 0 , 260 , 9 , 0 , 0 , 0 , 0 ,
1649 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 10 , 0 , 0 , 0 , 5 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 22 , 0 , 0 , 0 , 0 , 0 ,
0 , 0 , 0 , 3466 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 12 , 0 , 0 , 0 , 0 ,
132 , 0 , 0 , 0 , 0 , 0 , 0 , 40 , 41 , 0 , 0 , 0 , 0 , 0 , 76 , 0 , 0 , 0 0 , 0 , 0 , 4 , 0 , 87 , 17 , 0 , . . .

Each element represents a count of the specific system call requested. Because complete list of Android system calls is too large to show in this paper, so we only list a part of them.

Each number separated by commas, represents how many request/executions have been made by a specific Android application during the monitoring process. For example, the system call open() is used 25 times and kill() 47 times.

The architecture of the whole behavior-based malware detection cloud computing server is shown as Figure. 4.

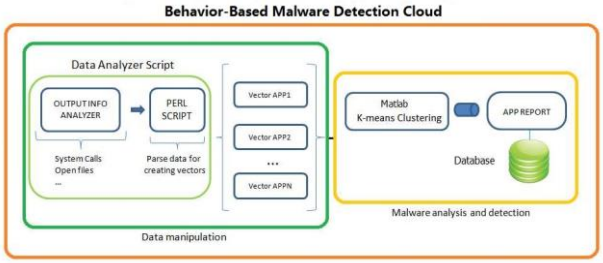

Figure 4: The architecture of behavior-based malware detection cloud computing server

5. Experimental results

The detailed results of the experiments carried out using our malware detection system are provided in this section. We have tested the whole system using two real malware specimens: PJApps contained in Steamy Window application. In this case, we obtained 100% detection accuracy for PJApp.

Steamy Window is a free application available at the Android Market. The malicious version of the application containing PJApps malware, which was discovered in unofficial repositories, six interactions were performed to test the system, 4 using the original Steamy Window application and 2 with malicious code of PJApps malware attached. System call feature vectors where collected and clustered with k-means algorithm.

The feature vectors of processed system calls collected during the different interactions of users with the application is shown as follows:

InteractionA= 0 , 0 , 0 , 3 , 7 , 7 , 7 , 0 , 0 , 1 , 1 , 0 , 0 , 11 , 0 , 1 , 0 , 0 , 0 , 3 ,

438 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 5 , 0 , 0 , 0 , 1 , 1 , 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0 , 0 ,
0 , 0 , 0 , 0 , 0 , 0 , 12 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ,
0 , 4 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 12 , 7 , 0 , 0 , 0 , 0 , 1 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ,
0 , 0 , 0 , 0 , 8 , 1 , 3 , 4065 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 2 , 0 , 0 , 0 , 0 , 2 , 2 , 0 ,
0 , 0 , 0 , 0 , 14011 , 0 , 0 , 0 , 0 , 0 , 648 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 6 , 0 , 0 , 0 , 0 , 0 ,
0 , 12 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ,
0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ,
InteractionB =0 , 0 , 0 , 34 , 43 ,4 5 , 87 , 0 , 0 ,5 ,5 ,0 ,0 ,4 7 , 0 , 5 , 0 ,
0 , 0 , 3 1 , 26 95 , 0 , 0 , 0 , 4 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 22 , 0 , 0 , 0 , 5 , 5 , 0 , 0 , 27 ,
0 , 0 , 0 , 46 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 48 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ,
0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 16 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 1 3 2 , 88 , 0 , 0 , 0 , 0 ,
2 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 60 , 5 , 27 , 13717 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ,
0 , 0 , 0 , 0 , 16 , 0 , 0 , 0 , 0 , 68 , 262 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 2328 , 0 , 0 ,
0 , 0 , 0 , 0 , 0 , 0 , 38 , 0 , 0 , 0 , 0 , 0 , 0 , 1 3 2 , 0 , 0 , 0 , 0 , 0 , 2 , 0 , 0 , 0 , 1 , 0 , 0 ,
0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ,
InteractionC =0 , 0 , 0 , 19 , 12 ,28 , 29 , 0 , 0 ,1 ,1 ,0 ,0 ,22 , 0 , 1 , 0 ,
0 , 0 , 19 , 1718 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 11 , 0 , 0 , 0 , 3 , 1 , 0 , 0 ,
4 , 0 , 0 , 0 , 8 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 36 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ,
0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 4 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 41 , 21 , 0 , 0 , 0 , 0 , 2 ,
0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 24 , 1 , 19 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 6 ,
0 , 0 , 0 , 0 , 27 , 15 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 1855 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ,
1 1 , 0 , 0 , 0 , 0 , 0 , 0 , 41 , 0 , 0 , 0 , 0 , 0 , 1 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ,
0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ,
InteractionD =0 , 0 , 0 , 16 , 12 ,27 , 28 , 0 , 0 ,1 ,1 ,0 ,0 ,19 , 0 , 1 , 0 ,
0 , 0 , 16 , 1214 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 8 , 0 , 0 , 0 , 2 , 1 , 0 , 0 , 4 , 0 ,
0 , 0 , 7 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 24 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ,
0 , 0 , 0 , 0 , 0 , 0 , 4 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 40 , 2 0 , 0 , 0 , 0 , 0 , 2 , 0 , 0 ,
0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 21 , 1 , 16 , 8597 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ,
6 , 0 , 0 , 0 , 0 , 27 , 15 , 0 , 0 , 0 , 0 , 0 , 29712 , 0 , 0 , 0 , 0 , 0 , 1549 , 0 , 0 , 0 ,
0 , 0 , 0 , 0 , 0 , 11 , 0 , 0 , 0 , 0 , 0 , 0 , 4 0 , 0 , 0 , 0 , 0 , 0 , 1 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ,
0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ,
InteractionE =0 ,0 , 0 , 48 , 73 ,67 ,139 , 0 , 0 ,8 ,8 , 0 , 0 , 56 , 0 , 8 ,
0 , 0 , 0 , 3 8 , 2964 , 0 , 0 , 0 , 8 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 28 , 0 , 0 , 0 , 6 , 8 , 0 , 0 ,
45 , 0 , 0 , 0 , 78 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 4 8 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ,
0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 24 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 210 , 151 , 0 , 0 ,
0 , 0 , 3 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 9 3 , 8 , 3 7 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ,
0 , 0 , 21 , 0 , 0 , 0 , 0 , 108 , 501 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 2328 , 0 , 0 , 0 ,
0 , 0 , 0 , 0 , 0 , 65 , 0 , 0 , 0 , 0 , 0 , 0 , 210 , 0 , 0 , 0 , 0 , 0 , 2 , 0 , 0 , 0 , 1 , 0 , 0 , 0 ,
0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ,
InteractionF =0 , 0 , 0 , 22 , 13 ,29 , 30 , 0 , 0 , 1 , 1 , 0 , 0 , 32 , 0 , 1 , 0 ,
0 , 0 , 22 , 2512 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 1 4 , 0 , 0 , 0 , 4 , 1 , 0 , 0 , 4 ,
0 , 0 , 0 , 12 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 4 8 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ,
0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 4 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 44 , 22 , 0 , 0 , 0 , 0 , 2 ,
0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 27 , 1 , 22 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 7 ,
0 , 0 , 0 , 0 , 28 , 15 , 0 , 0 , 0 , 0 , 0 , 48565 , 0 , 0 , 0 , 0 , 0 , 2328 , 0 , 0 , 0 , 0 ,
0 , 0 , 0 , 0 , 12 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 44 , 0 , 0 , 0 , 0 , 0 , 1 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ,
0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ,

The distance matrix between each interaction of Steamy Window applications with Euclidean distance as similarity metric are shown as Table 1.

Table 1: Euclidean distance between each interaction of Steamy Window applications

Interaction	A	B	C	D	E	F
A	0	0.1818	0.1414	0.1414	0.1818	0.1414
B	0.1818	0	0.1768	0.1768	0.1616	0.1667
C	0.1414	0.1768	0	0.1010	0.1818	0.1212
D	0.1414	0.1768	0.1010	0	0.1818	0.1212
E	0.1818	0.1616	0.1818	0.1818	0	0.1717
F	0.1414	0.1667	0.1212	0.1212	0.1717	0

Distances close to 0, are identical or similar vectors. Distances with a value far from 0, are non-similar vectors. The distance matrix shows that values for interactions B and E compared to the benign application interactions are higher than others.
Then, k-means algorithm has been used to cluster the interactions. Results are shown in Table 2.

Table 2: Steamy window clustering result

Interaction	A	B	C	D	E	F
Cluster	1	2	1	1	2	1
Application	benign	malware	benign	benign	malware	benign

In Table 2, First row shows the cluster number that each interaction is given as a result of applying k-means. This row contains the number of the cluster to which the data belongs. Second row shows which of the interactions were benign and malware. The system is able to correctly identify the two malicious applications, B and E, which is an indication that the behavior-based Android malware detection system is able to detect malicious executions of the Steamy Window application.

6. Conclusions

In this paper, we have proposed a new malware detection system to obtain and analyze smartphone application activity. It will be capable of distinguishing between benign and malicious applications of the same name and version, detecting anomalous behavior of known applications. We have indicated that monitoring system calls is a feasible way for detecting Android malware. The most important contribution of this work is the mechanism we propose for obtaining real traces of application behavior. Cloud computing helps the community to obtain real application traces of hundreds or even thousands of applications.

Acknowledgments

This work is supported by the Humanity and Social Science Youth foundation of Ministry of Education of China (No. 15YJC860001). This research is also supported by Social Science Foundation of Qingdao, China (No. QDSKL150437), China Postdoctoral Science Foundation Funded Project (No. 2015M580571) and the national training programs of innovation and entrepreneurship for undergraduates (No. 201511065006).

Reference

Asaf S., Moskovitch R., Elovici Y., Glezer C., 2009, Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey, Information Security Technical Report, 1, 14-22, DOI: 10.1186/2190-8532-1-1

Chengfeng, J., Wei, Z., Yue, Y., 2015, A micro-gesture recognition on the mobile web client, Review of Computer Engineer Studies, 2, 19-24, DOI: 10.18280/rces.020205

Damopoulos D., Kambourakis G., Gritzalis S., 2014, Exposing mobile malware from the inside (or what is your mobile app really doing?), Peer-to-Peer Networking and Applications, 4, 687-697, DOI: 10.1007/s12083-012-0179-x

Dagon D., Martin T., Starner T., 2004, Mobile phones as computing devices: The viruses are coming!, IEEE Pervasive Computing, 3, 11-15, DOI: 10.1109/MPRV.2004.21

Egele M., Scholte T., Kirda E., 2012, A survey on automated dynamic malware-analysis techniques and tools, ACM Computing Surveys, 2, 1-42, DOI: 10.1145/2089125.2089126

Gandotra E., Bansal D., Sofat S., 2014, Malware Analysis and Classification: A Survey, Journal of Information Security, 2, 56-64, DOI: 10.4236/jis.2014.52006

Jiaorao, L., Shanshan, Y., 2015, Time series analysis in the application research of the personal income tax planning in colleges and universities, Mathematical Modelling of Engineering Problems, 3, 1-4, DOI: 10.18280/mmep.020301

Rubanov V., Silakov D., 2014, Ensuring portability of Linux applications through standardization and knowledge base driven analysis, Science of Computer Programming, 2, 234-248, DOI: 10.1016/j.scico.2014.01.009

Shabtai, A. Fledel Y., Kanonov U., Elovici Y., Dolev S., Glezer C., 2010, Google android: A comprehensive security assessment, IEEE Security and Privacy, 1, 35-44, DOI: 10.1109/MSP.2010.2