Pervasive and Mobile Computing 24 (2015) 231–242

Contents lists available at ScienceDirect

Pervasive and Mobile Computing

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

Real-time and intelligent private data protection for the Android platform

Shih-Hao Hung *, Shuen-Wen Hsiao, Yu-Chi Teng, Roger Chien

Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, ROC

ARTICLE INFO

Article history:
Available online 3 September 2015

Keywords:
Android security
PasDroid
Mobile malware
Privacy
Information leakage

ABSTRACT

As the number of smart mobile devices and applications continue to grow dramatically, private data stored and handled by such mobile devices have become the primary targets of hackers and malicious software. Today, many malicious mobile applications steal user information, make premium calls, and send advertisement messages without the user's permission. Unfortunately, the Android system, currently the most popular smart mobile platform, only provides the users with a simple permission granting mechanism during the installation of applications, which are often ignored since most of the users do not pay attention to the potential hidden risks. Even though some vendors have integrated mechanisms to let the users grant or revoke the permissions associated with any applications at any time, such mechanisms are rarely used because the users do not know when, how and what sensitive information have been leaked.

In this paper, we proposed mechanisms to track the use of sensitive information by Android applications. We extended TaintDroid to build a real-time security framework, called PasDroid, with mechanisms to trace dubious data flow, map user–application interactions and alert the users about potential privacy leakage on the fly. The information provided by PasDroid enables the users to determine if a transmission should be allowed or blocked with intelligent security policies. Our experimental results show that PasDroid can be deployed with an affordable runtime overhead to help protect users against malicious applications. The design of security policies is key to eliminate false alarms and improve the user experience.

1. Introduction

Today's smart mobile platforms allow the users to install and perform applications, such as games, document editors, social network software and so on. The number of available applications has grown quickly [1], and the average number of applications installed on a smart phone had increased from 32 to 41 between 2011 and 2012 [2]. As the users install more applications on their phones, the higher chance that they would lose their private information or money to malicious software, a.k.a. *malware*.

A smartphone application may access the personal information of the user, including files, database, address book, SMS content, GPS location data, movement data by G-sensor and accelerometer. Some applications may even access the information stored in the cloud by the users. It is crucial to distinguish malware from legitimate (benign) applications

* Corresponding author.
E-mail addresses: hungsh@csie.ntu.edu.tw (S.-H. Hung), r00922122@csie.ntu.edu.tw (S.-W. Hsiao), r01922101@csie.ntu.edu.tw (Y.-C. Teng), roger.swchien@gmail.com (R. Chien).

http://dx.doi.org/10.1016/j.pmcj.2015.08.006
1574-1192/

to protect such personal information. Although *Google Play*, the largest application store for the Android platform, has an application review mechanism called *Bouncer* [3], the number of Android malware continues to increase dramatically [4].

The current Android operating system only prompts the user to review and grant the *permissions* requested by an application during the installation. When version 4.3 of Android was released, a built-in hidden feature called *App Ops* allowed the user to revoke some permissions from an installed application, but this feature was changed to be inaccessible to the user when version 4.4.2 was released [5]. Some vendors have modified their system images and provided utilities which are similar to App Ops. Even with such modifications, it is still difficult for the users to control the permissions because most of the users do not know when, how, and what information is being accessed by the applications. For example, if a user allows an application to access the address book and the Internet, there is no way for the user to ensure if the application will not send the contents in the address book to untrusted parties on the Internet. An SMS spam malware can send out the user contact list without letting the user know its real intention.

To address this issue, it is necessary for the user to know which pieces of private sensitive data an application has accesses and where any information derived from the sensitive data have been delivered to, so that the user can decide whether to grant the network access or not based on such knowledge. Thus, we created a security enhancing mechanism, called *PasDroid*, to protect the privacy of the users of Android phones by revealing the information flow of sensitive data and enabling the users to stop a potential information leakage in time [6]. (The prefix "Pas" is the abbreviation of the name of our laboratory, "Performance, applications, and security".) This paper further extends our previous work to map user–application interactions and handle the potential information leakage with an intelligent scheme based on a two-level taint policy design.

The major contribution for this work is the following. (1) We leveraged an existing work, *TaintDroid* [7], and extended it to cover more source data types, such as files on SD card, and capture any suspicious outgoing message before sending it out. (2) We added a new mechanism to keep track the interactions between an application and the input events, and add the intelligent mechanism to classify if a data transmission is highly related to the user intention. (3) We added a control mechanism for the user to let the intelligent classifier should block outgoing messages automatically or handle the situation manually to skip frequent false alarms.

The reminder of this paper discusses how the enhanced PasDroid framework is designed and implemented, with experimental results to show its usefulness and effectiveness. Section 2 describes the related works regarding to the enforcement of security and privacy for the Android platform. Section 3 further mentions some background knowledge on the internals of the Android platform for the development and implementation of PasDroid. Section 4 discusses the design of the PasDroid framework. Section 5, presents case studies and experimental results to validate the proposed framework and to evaluate its overhead. Finally, Section 6 concludes this paper and points out some future directions for further improving PasDroid.

2. Related works

The permission control system in the Android platform can be regarded as *coarse-grained*, since each granted permission gives an application the privilege to access a collection of resources without requiring the application to declare to the user how the resources will be used in detail. To address this weakness, several approaches have been proposed to amend the Android permission system by specifying or changing the policies during the runtime.

Saint [8] is a run-time application management system for Android, which aims at protecting applications from each other by enforcing a set of application policies, so that an application can specify which applications can access its interfaces and how other applications use these interfaces. *Apex* [9] and *CRePE* [10] both include security extensions to support context-related policy enforcement at the runtime. *Bai* [11] has further extended the Android security model to support part of the UCON model, which is based on *usage control*. All these previously proposed extensions address some security limitations, but none of them actively prevent applications from leaking data.

MockDroid [12] introduces a system which can limit the access of the installed applications to specific data areas (e.g. SMS, device ID, location, contacts, etc.) during the runtime. For example, even if a game is granted with the permissions to access the GPS location and the network, the user can still disable the network access for the game when the user does not think that the game needs to connect to the network and re-enable the network access when it is time to upload game score or download extra data from the server. However, it is tedious for the user to control such a permission system this way, and a malicious application can still lure the user to turn on the network connection and send out sensitive information together with a seemingly reasonable purpose.

TISSA [13] provides users with the ability to define the *privacy level* for the information which are revealed to an application. Four levels of privacy are introduced: *trusted*, *empty*, *anonymized*, and *bogus*, such that the user can specify how the application reads sensitive data according to the privacy level. The application acquires the actual data, empty data, anonymized data, or fake data, depending on its privacy level. Still, the user cannot stop a trusted application from sending the data to untrusted parties.

A *privilege spreading attack* [14] occurs when an unprivileged application exploits the system to gain the permissions from privileged applications. Designing colluding applications to purposely provide permissions to other applications has become popular [15,16]. A malicious application may exploit the vulnerability in a benign application, which is often referred to as a *confused deputy attack*. *Quire* [17] provides a lightweight provenance system that prevents such attacks by tracking the RPC