IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Volume-03, Issue-09 E-ISSN: 2347-2693

To develop an application in android for smart phones tasks processing to the cloud to detect malware in application and generate reports

Sangita Mahajan*[1], Suvarna Sangle[2] and Gayatri Khairnar[3]

[1*,2,3]Department, Of Computer Engineering, Savitribai Phule Pune University, Maharashtra, India

www.ijcseonline.org

Received: Aug/12/2015 Revised: Aug/28/2015 Accepted: Sep/20/2015 Published: Sep/30/2015

Abstract**—** Most of us use android phones these days and also uses the multiple applications and facilities frequently. Play store provides great number of application but unfortunately few of those applications are fraud. Such applications dose damage to phone and also possibly data thefts. Hence such applications must be marked, so that they will be recognizable for play store users. So we are proposing a web application which will process the information, comments and the reviews of the application on cloud server. As we are handling the big data here so the process is done on cloud server and malware is detected. So it will be easier to decide which application is fraud or not. Multiple applications can be processed at a time with the web application.

Keywords**—** Android, Permissions, Security, Instrumentation, Privacy, Risk assessment

I. INTRODUCTION

There has been a steady rise of smart mobile devices for both personal and business use. These devices run applications that have unmatched access to private information, including contacts, emails, geo-localization data, personal and business files, and much more. There is also explicit monetary risk integrated with these devices since phone calls, messages, and data usage can cost money. More directly, these devices often have access to users' bank accounts through an application or as a means to authenticate to a bank, and in the future it seems likely that phones may act as a digitize notecase, directly accessing the bank accounts as part of the range of capabilities**.** While the existence of this information and access creates much of the value found in a android mobile device, it also makes these devices proposing targets for malicious entities.

The prototype for program distribution on these mobile devices also differs from that of the traditional PCs. Many developers are releasing applications to one or a few centric application markets. While there are third party application stores, currently all popular mobile device platform have centric application stores as the primary mechanism of application distribution. Smart phones has Google Play as the primary store, Kindle uses the Amazon App store for Android, iOS has iTunes App Store, Windows RT has a Windows Store, and BlackBerry has App World. This new paradigm presents both challenges and chance for malware defense. Instead of most programs coming from a relatively small number of reputable vendors, which enables protection based on white listing and signed software distributions, in mobile devices there are many more developers, many of which have insufficient history to establish reputation. On the other hand, centralized markets provide chance for techniques to analyze applications by extracting some set of measurable features, and identifying potentially malicious applications in the set. In researchers have developed several approaches that use the permissions requested by an Android application to identify whether the application is potentially malicious. In requesting a certain permission or a certain combination of two or three permissions triggers a warning that the application is risky. In requesting a critical permission that is rarely requested is view as a signal that the application is risky. In four probabilistic generative models are used to identify potentially malicious applications include Basic Naive Bayes (BNB), Naive Bayes with informative Priors (PNB), Mixture of Naive Bayes (MNB), Hierarchical Mixture of Naïve Bayes (HMNB). Experimental results show that these models significantly outperform prior approaches in using the area under curve (AUC) for the receiver operating characteristic (ROC) curve as evaluation metric.

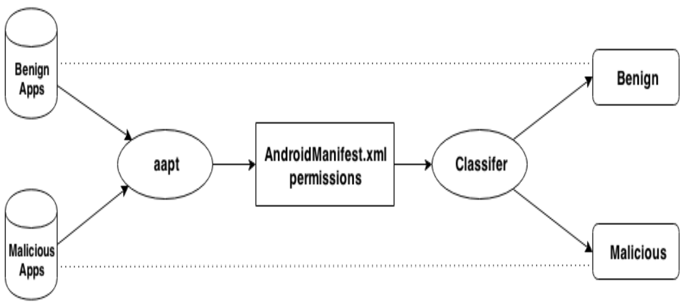

Fig.1. To create the classifier's dataset

The permission declare in the AndroidManifest.xml le of an application are automatically extracted using the Android Asset Packaging Tool (adapt). Then, the classier automatically labels the application behavior, as either benign or (potentially) malicious, according to the combination of permissions the application requires.

II. LETURATURE SURVAY

Most of us use android phones these days and also uses the play store facility frequently. Play store provides great number of application but unfortunately few of those applications are fraud. Such applications dose damage to phone and also possibly data thefts. Hence such applications must be marked, so that they will be recognizable for play store users.

Ask for a certain permissions or a certain combination of two or three permissions triggers a warning that the applications are risky. The author suggest that researchers have developed several approaches that use the permissions requested by an Android application to identify whether the application is may be malicious[1].

Ask for a critical permission that is rarely requested is view as a signal that the application is risky. By using the area under curve (AUC) for the receiver operating characteristics (ROC) curve as assessment metric[2].

Generative probabilistic models in assume that some parameterized random process generates the application data (i.e., permissions) and learn model parameters which optimize the fit of the model to the applications used in training. Then one can compute the may be of each application being generated by the models, and identify those with low probabilities as potentially malicious applications. The strength of generative models is that they work with unlabeled data, in other words without information on malicious applications[3].

In the context of smart phones some work on static analysis has also been performed, typically focusing on all functions that are used by an applications. One common problem in third party application stores is piracy and malware. It is possible to buy an applications, repackage the applications unchanged or with malicious code added, and then submit it to a third party applications store for others to install. Desno s[5] uses compression and distance on source code to find applications that have high overlap in order to detect theft and malware between the official market and third party markets. Schmidt et al use static function call analysisto detect mobile malware.

Provides a framework to dynamically analyze application behavior to detect malware on the Android platform. They collect the system traces from many real users and send them to a central server for analysis to detect behavioral differences between applications that should generally have the same behavior[6]. Their goal is primarily to detect malware that has been repackaged and distributed on third party application stores.

III. PROPOSED SYSTEM

System collects all the information about particular application such as any comment or reviews about it. We are sending this information and .apk file of that application to cloud server i.e. taking it to process it. This will process for detecting any malwares in application and give report. Hence we can detect the application if it is fraud or not.

A) Application analysis and repackaging

As author described in the introduction we are interested in defining a method allowing to effectively profile and analyzing mobile application in search for over-privileges. The approach adopted is based on application combination to verify the real need of requesting and taking access to all the “sensitive” Android's APIs. This approach requires neither access to the application original source code nor modified by the underlying framework [4]. We depend on static analysis to compute the set of permission that might be used by the examined application, and on dynamic analysis to upgrade their use. The outputs of both static and dynamic analysis are combined and compared with the manifest's permission set to deduce whether the applications is over-privileged or not.

B) Theoretical framework

In this section author present the theoretical framework on which we based our approach. An application is considered over-privileged if and only if there is a permission object in the manifest set (Mp) which is not listed in the static analysis permissions set (Sp) as illustrated in Equation. The over-privilege set is computed as the intersection between the Mp and Sp as illustrated in Equation . Complementary, an application is marked as non over-privileged if and only if the static, dynamic and manifest permissions sets match as illustrated in Equation. These cases are graphically illustrated in as well. In the case where the manifest (Mp) and the static (Sp) permissions sets are same, but they are a super set of the dynamic (Dp) permissions set then a specific set of the examined application's permissions, according to Equation might be susceptible to over-privileged flaws with a certain confidence level.

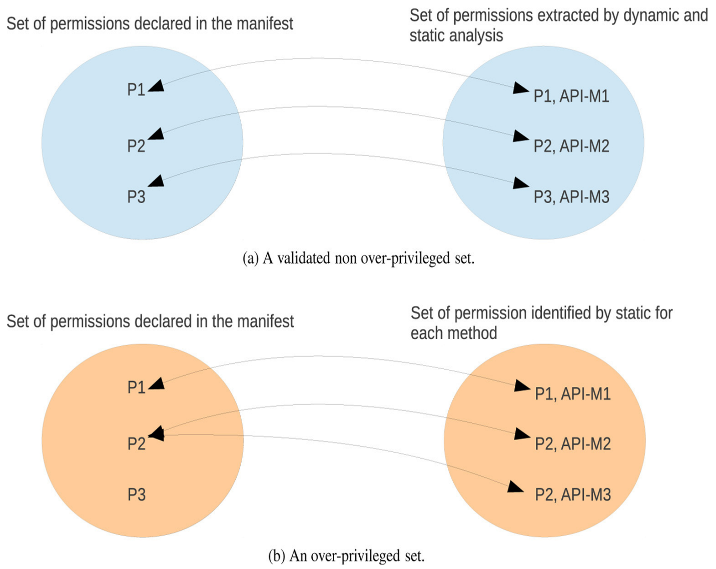
Set of permissions declared in the manifest
Set of permissions extracted by dynamic and static analysis
P1
P2
P3
P1, API-M1
P2, API-M2
P3, API-M3
(a) A validated non over-privileged set.
Set of permissions declared in the manifest
Set of permission identified by static for each method
P1
P2
P3
P1, API-M1
P2, API-M2
P2, API-M3
(b) An over-privileged set.

Fig. 2 Definition of validated non over-privileged and over-privileged application

C) Permissions' risk & false identification assessment

To assess possible false identification that our approach might generate and measure the exposure of the examined application to over-privileged threats we depends on the attack surface introduced by the set of privilege not reached during the dynamic analysis phase, where S corresponds to the percentage of the permissions triggered during the execution according to It should be mentioned also that if the static analysis set is a super set of the manifest one, we analyze only the manifest set. This is because Android OS will throw a security exceptions according to its security design, for any sensitive API having no declaration of the permissions needed for its execution in the application manifest. Table 1 summarizes the risk assessment class provided by the proposed framework.

D) Identify over-privileged applications

The outcomes of the analysis extracts the manifest permissions set and generate a new Android mobile applications package, the instrumented one which records all the possible methods that exist in the reversed engineer applications and the methods executed at run-time. Note that the reversed engineer code does not necessarily correspond to the application original source code however
the API calls will be the same to the original code. The instrumented applications can be executed either manually or automatically depending on the Android-monkey test suite which is able to generate and inject to the examine application pseudo-random stream of user events in a random but repeated test cases. The Monkey test suite is an official Android frameworks that can be used by developers for checking application's robustness before published it. we reproduce common real life situation and record the execute method. In case that additional input are available e.g when users' exercise the applications they can be inserted through the virtual device drivers available in Android When the execution is complete created the method permissions map for both the static and the dynamic analysis.

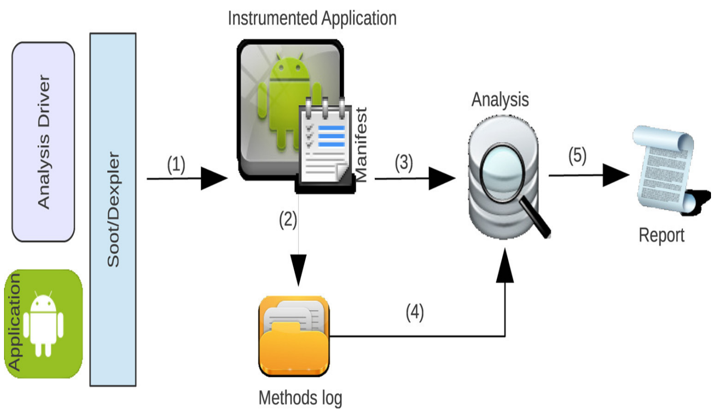

Fig.3. Proposed permission verification approach.

The soot framework analyzes the mobile application and inserts small pieces of code to monitor the executed methods. Every time the application is executed all the methods are logged and analyzed in order to determine whether the application uses all the requested permissions, according to the proposed theoretical framework.

IV. SYSTEM ARCHITECTURE

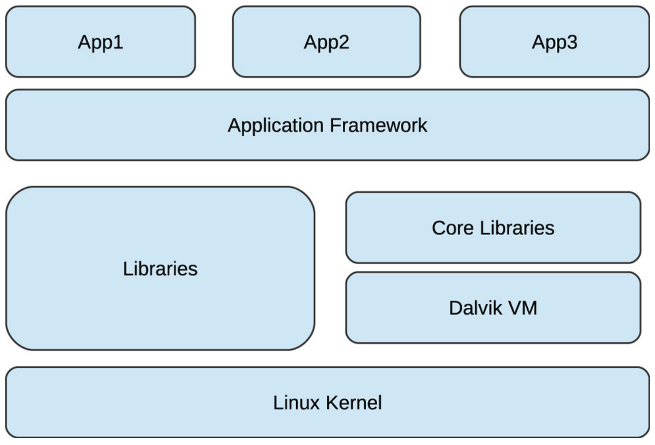

Fig. 4 Android software modular architecture

The core of the Android Operating System is built on top of the Linux kernel. This enables it to provide strong isolation for protected user data system resources and avoiding conflicts, for both Java programming language and native Android mobile application. Overview the Android OS architecture. The Android OS system runs each application under the privileges of different "users" and assigns a unique user ID to every user. This approach differs from other Operating system where multiple applications run under the same user permission. By default application are not allowed to execute functions that might affect other applications or users and they have access to a limited set of resource. Application must mandatory declare in a manifest all the "sensitive" operations that can take place in the course of execution the users during the installation process they are requested to endorse them otherwise the installation fails. In case an application executes a protected feature that hasn't been declared in the manifest a security exception will throw during execution.

V. CONCLUSION

Over-privileged applications introduce new chance for Manipulating personal information and sensitive functionalities managed in android phones. Users have to trust applications requests for accessing sensitive resources, defined in their manifests, if they would like to install and also use them. However, even benevolent over-privileged applications might be utilized by malicious applications to grant access to otherwise not-accessible data. In this paper, we present Android mobile application permissions risk assessment approach that combines static and dynamic analysis to assess any given application as over-privileged with definite degree of probability. This approach not only can accurately identify whether an application is over-privileged with certain confidence level but also validates the need of requesting access to the permissions declare in application's manifest. Our approach is orthogonal to other solutions, and can be used in order to compute mobile applications attack surface and the risk presented by over-privileges.

VI. REFERENCES

[1] W. Enck, M. Ongtang, and P. McDaniel, "On lightweight mobile phone application certification," in Proc. 16th ACM Conf. Comput. Commun. Security, **2009**, pp. 235–245.

[2] B. P. Sarma, N. Li, C. Gates, R. Potharaju, C. Nita-Rotaru, and I. Molloy, "Android permissions: A perspective combining risks and benefits," in Proc. 17th ACM Symp. Access Control Models Technol., **2012**, pp. 13–22.

[3] H. Peng, C. Gates, B. Sarma, N. Li, Y. Qi, R. Potharaju, C. Nita- Rotaru, and I. Molloy, "Using probabilistic generative models for ranking risks of Android apps," in Proc. ACM Conf. Comput. Commun. Security, **2012**, pp. 241–252.

[4] Ali K, Lhot_ak O. Application-only call graph construction. In:Proceedings of the 26th European onference onObject-Oriented Programming. Springer-Verlag; **2012**.p. 688e712. "A tool for reverse engineering android apk files."

[5] A.-D. Schmidt, R. Bye, H.-G. Schmidt, J. Clausen, O. Kiraz, K. A. Yuksel, S. A. Camtepe, and S. Albayrak, "Static analysis of executables for collaborative malware detection on Android," in Proc. IEEE Int. Conf. Commun., **2009**, pp. 1–5.

[6] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: Behavior-based malware detection system for Android," in Proc. 1st ACM Workshop Security Privacy Smartphones Mobile Devices, **2011**, pp. 15–26.