DefDroid: Securing Android with Fine-Grained Security Policy

Chao Huang
School of Software
Shanghai Jiao Tong University
Shanghai, China
bujingyun_beta@sjtu.edu.cn

Shuohong Wang
School of Computer Science
Fudan University
Shanghai, China
sh_wang@fudan.edu.cn

Haiyang Sun
Faculty of Informatics
Università della Svizzera italiana
Lugano, Switzerland
haiyang.sun@usi.ch

Zhengwei Qi
School of Software
Shanghai Jiao Tong University
Shanghai, China
qizhwei@sjtu.edu.cn

***Abstract*—Android occupies the absolute dominant position in mobile operating system and has the largest market share. Meanwhile, Android faces the risk of malicious insiders leaking sensitive information. In this paper, we present DefDroid, a repackaging tool for enforcing security policies by modifying Android applications without root privilege. The main advantages of DefDroid are that it provides a user-friendly interface to configure fine-grained policies and it supplies multiple deployment methods. We have implemented policies aimed at three types of services of Android system, i.e., content provider, file system, and network. We choose 74 arbitrary applications from Android market and the experimental results show that the successful rate of repackaging applications is about 94.6% which effectively improve the privacy security of Android system while the increased overhead can be tolerated.**

***Keywords*—Android; permission restriction; repackage; bytecode instrumentation**

I. Introduction

Android OS (developed by Google) spreads around the world and becomes one of the most important mobile operating systems. According to data from the International Data Corporation (IDC), Android occupies 84.4% of Smartphone OS Market Share in the second quarter of 2014 [1]. There are more than 1 billion Android devices activated around the global and Android is the dominant mobile OS in the world [2].

According to the report of The Wall Street Journal [3], antivirus software only catches 45% of malware attacks and 55% malware behaviors are missed. In Android ecosystem, sensitive information leakage is a serious problem due to the rapid expansion of Android mobile phone customers. The native Android permission system follows an 'all or nothing' policy to restrict applications' access to privileged resources. That is to say, an application cannot be successfully installed unless all permissions it applies for are allowed. This coarse-grained access control system brings terrible security problem and privacy leakage.

To overcome the issue, companies and researchers come up with several methods [4]–[6] to provide fine-grained security policy for Android which can be divided into three types. The first approach is native library rewriting which imposes fine-grained access control by intercepting the native calls. However, it does not allow security administrator to write and deploy unified security policies according to different requirements of companies. The second one is fine-grained access control that can be achieved by modifying the Android operating system. The defect of it lies in that it demands root privilege and it is unable to get more high-level, unformed information. The last approach is based on bytecode instrumentation technique, which makes it possible to obtain detailed information of application behaviors at Java level and protect sensitive information at greater extent. However, there is no satisfactory method based on this technique (to our knowledge). Consequently, we propose a novel policy enforcement system using this technique.

Our Approach To protect sensitive information of Android system, we propose a novel and deployable tool with fine-grained security policy. DefDroid does not need root privilege, which avoids modifying the Android operating system. On the contrary, security policies provided by DefDroid are implemented into arbitrary applications by repackaging applications itself. The repackaging applications can be installed on users' mobile devices after signing them with valid keys. The policies provided by DefDroid are implemented into applications through bytecode instrumentation. DefDroid translates user-defined security policies into instrumentation code and inserts these codes into specific code snippets to enforce security.

Theoretically, DefDroid is able to monitor all interaction with Android operating system except invocation of native code. DefDroid enables much more fine-grained policy enforcement than original permission model of Android and other solutions, such as FireDroid [4] and Aurasium [5]. There are three groups of policies and each group represents a service of Android operating system, i.e., content provider, file system, and network.

The main contributions of this paper are that

- We develop fine-grained policies to protect different types of sensitive information in Android system, such as protection of specific contacts and sandbox of file system.
- We provide groups of policies to customers so that users can customize policies to protect their sensitive information without knowing too much about implementation details.
- We show a workable solution to deal with bytecode instrumentation and repackaging applications.

DOI reference number: 10.18293/SEKE2015-162

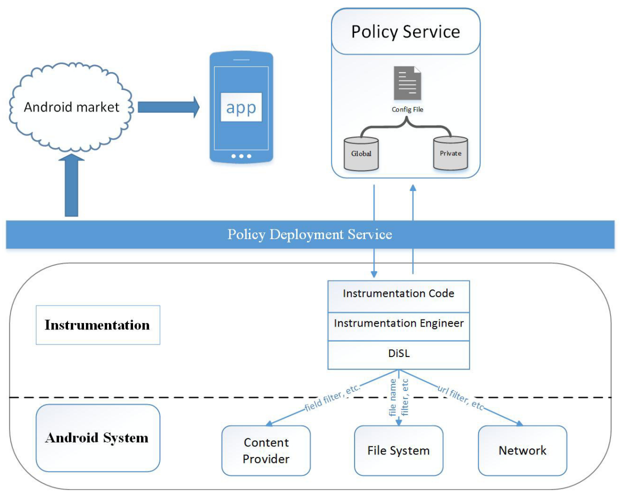

Fig. 1. Architecture of DefDroid

- We propose a scheme for companies who need to manage information security of multiple users' mobile device.

II. DefDroid System Design

The main idea of our proposal is that modifying source code of Android applications can completely enforce fine-grained permission policies and block malicious behaviors, although there are several problems need to be solved.

DefDroid system is made up of several components, as shown in Fig. 1. In order to implement security policies, a convenient way to define these policies is necessary. So, we create a configure file to edit and store security policies. The Policy Service (PS) is made up of configure files. All security policies written in the files are divided into two parts, namely Global policies and Private policies. The Global policies store security policies that influence all applications in private Android market. Private Android Market (PAM) is an Android application download platform inside the company or non-public Android market that serves specific customers. Applications in private Android market are repackaged and inserted relevant code. Policy Deployment Service (PDS) is the middle component of DefDroid. PDS gets configure file of PS and parses it to related code. Then, PDS transmits this policy information to Instrumentation component. Also, PDS takes the task of publishing Android package file to PAM. More specifically, PDS reads modified 'classes.dex' file, and other related files and compresses them to Android package file. Bottom layer is made up of Instrumentation part and Android System part. Instrumentation component contains instrumentation code, instrumentation engineer (IE), and DiSL. Instrumentation code is Java code based on grammatical rules of DiSL converted from PDS. IE is the core module to implement bytecode instrumentation. IE compiles instrumentation code to Java bytecode and inserts it into specific code snippets. The whole policies are divided into three parts, i.e., content provider, file system and network. Content provider is the standard interface that connects data in one process with code running in another process [7]. A lot of sensitive information must be accessed through content provider, such as contact list, sms information, and media data. Policies aimed at content provider can effectively control information leakage in mobile phone. File system manages files and data stored in Android operating system. Policies of file system are able to protect data of storage devices, especially non-encrypted sensitive documents. Network is an important part of sensitive information protection. In several scenarios, some URL addresses are not allowed to be accessed because of security issues. Instrumentation of network service is able to enforce relevant policies.

III. Implementation Details

In this section, we introduce the implementation details of DefDroid. Different from FireDroid [4] which monitors applications' interactions with the OS, DefDroid implements fine-grained permission policies of sensitive resource by bytecode instrumentation. Other related work, such as Aurasium [5] is proposed to solve it by replacing Android system libraries.

We use DiSL as our instrumentation tool. DiSL (domain-specific language for instrumentation) [8] is a domain-specific language especially designed for dynamic program analysis and is primarily designed to manipulate and transform bytecode. We decompress the Android package file and get 'classes.dex' file and invoke several functions of dex2jar [9] to convert the file to 'classes.jar'. After instrumentation is finished, we use Android dx tool [10] to convert jar file to dex file. Finally, we compress all related files into Android package file and sign applications with users' private keys before publishing to Android market.

For deploying fine-grained policies, there are two alternatives adapted in DefDroid. One is static policies. This means relevant instrumentation codes are going to carry out these policies. The other is dynamic policies. The approach will insert some codes in the application, these codes are able to get specific operations on sensitive resources from the server. Policy makers can rewrite data in the server to deploy security policies rapidly.

The configure file that contains policy setting is an xml format file, as shown in Fig. 2. We develop a component in PDS to parse the configure file. The 'Global' tag and 'Private' tag determine the scope of policies, and the 'name' attribute of 'Private' tag is the name of affected application. The next level contains three kinds of tags, i.e., 'ContentProvider', 'FileSystem', and 'Network'. Policies of DefDroid are mainly aimed to harden protection of sensitive resource of these three parts.

A. *Content provider*

Content provider is the standard interface to manage sensitive data, such as audio, images, and personal contact information. So, if we monitor 'ContentResolver' object, we can monitor all operations aimed at content providers. The Contacts provider, one of content providers, stores all information about contacts of mobile phones. DefDroid provides contacts

```
<?xml version="1.0" encoding="UTF-8"?>
<Global>
  <ContentProvider>
    <Contacts groupName="Company"
        operation="all">false<Contacts>
  </ContentProvider>
</Global>
<Private name="applicationName">
  <FileSystem>
    <File filePath="file path"
        operation="read">true<File>
    <File extension="doc"
        operation="write">false<File>
  </FileSystem>
  <NetworkSystem>
    <network url="www.baidu.com" >false<Contacts>
  </NetworkSystem>
</Private>
```

Fig. 2. Example of configure file.

sandbox for specific fields. For example, most contacts have their own grouping, such as home, friend, and company, and DefDroid can block operations of contacts in specific group, as shown in Fig. 2. After being deployed, applications that have access to contact list are only enforced to operate on contacts in the 'company' group and other contacts are invisible.

B. File System

File system of Android operating system is similar to Linux system. Android has external storage and it's necessary to add permissions for write/read access to external storage in AndroidManifest.xml. It is ubiquitous that applications have access to external storage, however, users who use their mobile phones frequently are more likely to store critical files in external storage of mobile devices. Hence, external storage is not safe.

DefDroid is able to manage files in external storage, as shown in Fig. 2. The policy means files with 'doc' extension are unwritable for applications. The value of 'filePath' will be appended to root path of external storage to get the absolute path of the file. And the policy will be aborted when the file is not found in the absolute path.

C. Network

Network is the most complicated interface in Android framework. Malware and virus widely exist in network. Building website blacklist to block website access is a workable way to protect mobile device and improve the security.

DefDroid is able to manage network access through URL address in external storage, as shown in Fig. 2. The 'url' attribute of 'network' tag stores website address and the policy means that accessing website by the URL are forbidden.

IV. Performance Evaluation

In this section, we present the performance evaluation result of DefDroid. We implement security policies presented in Section III. We evaluate repackage performance and execution time before and after instrumentation. Our evaluation is conducted on a Google Nexus 7 Tablet running Android 4.4.2.

TABLE I. Evaluation results of repackaging

Type of App	Total	Repackaged	Success Rate
content provider	37	35	94.6%
file system	22	21	95.5%
network	15	14	93.3%

We download 74 applications from Android market, the functions of these applications correspond to policy model of DefDroid as introduced in Section III. There are 37 contacts applications, 22 file-related applications, and 15 network-related applications in total. Different applications are enforced different types of policies. For contacts applications, the policy is designed so that applications are only allowed to operate on contacts in 'Company' group. For file-related applications, we enforce a policy to prohibit write access of the applications to 'data.txt' file on the external storage. For network-related ones, applications are forbidden to visit website of "www.baidu.com" according to the policy.

As shown in Table I, there are altogether 4 applications failed to be repackaged. The reason is that the translator tool dex2jar still has several bugs resulting in several bytecodes translated from Dalvik bytecode not recognized by DiSL. Considering that the probability of this circumstance is very small, we ignore these questionable applications in our test set.

The experimental results of repackaging contacts applications, file-related applications and network related applications are shown as histograms in Fig. 3(a), 3(b), 3(c) respectively. The blue bins show the number of snippets modified by bytecode instrumentation (the left vertical axis displays their values) and the red ones show the execution time of repackaging each application (the right vertical axis displays their values). And the numbers on horizontal axis list each application of the corresponding type.

As shown in Table II, we measure the execution time of original and modified code snippets. The experimental results present the increased overhead caused by policies of DefDroid. For contacts, we need to query contacts table in database multiple times to get group id by group name and check if the contacts required by the applications are in this group. It definitely increases execution time compared with original code, and therefore we record these information of group id to avoid querying database repeatedly when requiring multiple contacts. According to the experimental results and user experience of modified applications, the overhead increases but can be tolerated. For policies of file system and network, we provide two modes of policy deployment, which are local and network respectively. Applications are forced to obtain data from Internet when implementation information is stored on servers, which will substantially increase execution costs. Therefore, it is recommended to store implementation information locally.

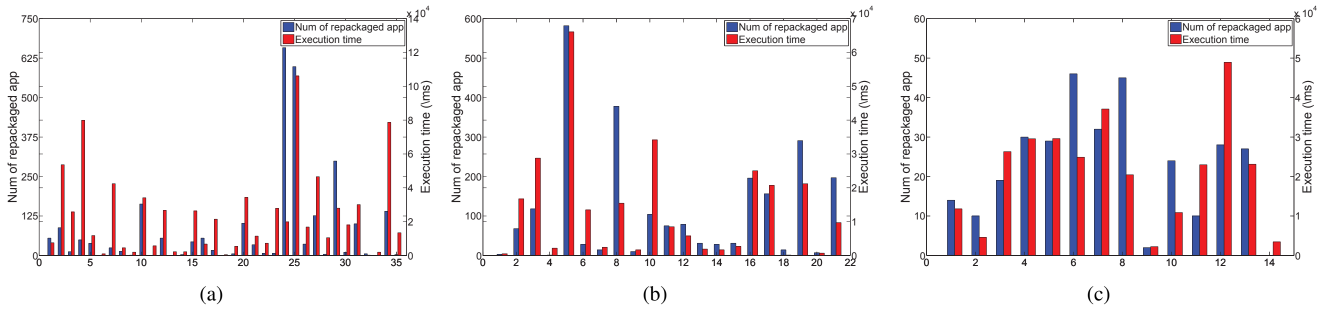

Fig. 3. Evaluation results of repackaging applications. (a). Evaluation results of repackaging contacts applications. (b). Evaluation results of repackaging file-related applications. (c). Evaluation results of repackaging network-related applications.

TABLE II. Overhead measurements of DefDroid's Policies

	Original(/ms)	DefDroid(/ms)	Overhead
Group field sandbox of contacts	9	23	60.9%
File system restriction	2	2	0%
		157	98.7%
URL restriction of network	168	168	0%
		318	47.2%

V. Related Work

In this section, we present an overview of the related work that aimed to deal with information leakage and malware detection.

Programing analysis has been applied to detect malicious applications in Android market. TaintDroid [11] is proposed and obtains good experiment results by employing dynamic taint analysis. Malware detection is able to clean malware applications in Android market. But malware that has already been installed in mobile devices are still doing malicious attacks.

Modifying Android application to enforce permission control policies through bytecode instrumentation is another approach to protect sensitive information. Aurasium [5] enforces policies by replacing Android's standard C libraries with Aurasium native libraries and bytecode instrumentation,

Ptrace is a tool used to track processes and modify system call. FireDroid [4] provides a fine-grained policy model by monitoring Android process with ptrace. Meanwhile, FireDroid contains a policy language to simplify policy making process. However, deploying FireDroid system is a difficult task, considering the high risk of rooting Android system.

VI. Conclusion

In this paper, we present DefDroid, a novel fine-grained permission control solution including policy server and application repackaging tool, which is able to act as defender of malware and malicious behaviors of applications for billions of Android mobile platform users. We implement DefDroid without rooting device, which reduces the potential risk of sensitive information leakage and is more acceptable to customers. The experimental results show that DefDroid increases overhead of applications compared with the original version. However, considering the protection of sensitive information that DefDroid provides, the performance reduction can be tolerated.

VII. Acknowledgment

This work is supported by NSFC (No. 61272101), National R&D Infrastructure and Facility Development Program (No. 2013FY111900), and NRF Singapore CREATE Program E2S2. We thank support from Shanghai Key Laboratory of Scalable Computing and Systems for this research.

References

[1] *Smartphone OS Market Share, Q3 2014*. [Online]. Available: http://www.idc.com/prodserv/smartphone-os-market-share.jsp

[2] *Android still the dominant mobile OS with 1 billion active users*. [Online]. Available: http://www.engadget.com/2014/06/25/google-io-2014-by-the-numbers/

[3] *Symantec Develops New Attack on Cyberhacking*. [Online]. Available: http://www.wsj.com/news/articles/SB10001424052 702303417104579542140235850578

[4] G. Russello, A. B. Jimenez, H. Naderi, and W. van der Mark, "Firedroid: hardening security in almost-stock android," in *Proceedings of the 29th Annual Computer Security Applications Conference*. ACM, 2013, pp. 319–328.

[5] R. Xu, H. Saïdi, and R. Anderson, "Aurasium: Practical policy enforcement for android applications," in *Proceedings of the 21st USENIX Conference on Security Symposium*, ser. Security'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 27–27.

[6] A. Bartel, J. Klein, M. Monperrus, K. Allix, and Y. L. Traon, "Improving privacy on android smartphones through in-vivo bytecode instrumentation," *arXiv preprint arXiv:1208.4536*, 2012.

[7] *Content Providers*. [Online]. Available: http://developer.android.com/guide/topics/providers/content-providers.html

[8] L. Marek, A. Villazón, Y. Zheng, D. Ansaloni, W. Binder, and Z. Qi, "Disl: a domain-specific language for bytecode instrumentation," in *Proceedings of the 11th annual international conference on Aspect-oriented Software Development*. ACM, 2012, pp. 239–250.

[9] *Dex2jar*. [Online]. Available: https://code.google.com/p/dex2jar/

[10] *Android apktool*. [Online]. Available: http://code.google.com/p/android-apktool/

[11] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'10. 2010, pp. 1–6.

[12] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall, "These aren't the droids you're looking for: retrofitting android to protect data from imperious applications," in *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 2011, pp. 639–652.