Context Awareness through Reasoning on Private Analysis for Android Application

Hongmu Han *,Ruixuan Li †,Jinwei Hu ‡,Meikang Qiu §
*School Of Computer Science & Technology,
Huazhong University of Science and Technology,
Wuhan, 430074, China
E-mail: hanhongmu@hust.edu.cn
† School Of Computer Science & Technology,
Huazhong University of Science and Technology,
Wuhan, 430074, China
E-mail: rxli@hust.edu.cn
‡ Department of Computer Science,
TU Darmstadt,
D-64289 Darmstadt, Germany
hu@mais.informatik.tu-darmstadt.de
§Department of Computer Science,
Pace University,
New York, USA
Email: mqiu@pace.edu

***Abstract*—Smartphone contains diverse sensitive private information, such as phone number, contact and credit. The Android platform employs permission mechanism to protect intensive information from illegal leakage. There are several tools to assist users to manage apps permissions, such as App Ops, Privacy Guard, and XPrivacy. However, the permission mechanism of Android is complicated, and only a few of smartphone users are familiar with Android security mechanism. In addition, it is difficult for smartphone users to know whether the privacy information exposure by Android applications is legal or not. Reverse engineering and program data flow analysis are basic approaches to analysis applications. Although it needs analysts to manual analysis results, its efficiency is largely depends on analysts' skills and experience. To improve analysis efficiency, we take a case-based reasoning method to simulation the security expert analysis applications. We translate the result of Android static analysis tools into cases and build a case reasoning library. Experiments show that it can help analysts to aid analysis Android applications.**

Keywords—Android; privacy leakage; static analysis; security; case based reasoning.

I. Introduction

According to the recent survey in Q1 2014, 279.4 million smartphones are shipped. 81% of these phones are running Android, 16% and 3% are running with iOS and Windows Phone respectively. As Android is becoming the most popular mobile operating system, the Android mobile devices are becoming a worth target for security and privacy violations. Smartphones are privacy rich devices. An app can get any required sensitive information with property privileges. Not only malicious apps, but also benign apps are accessing sensitive information, such as contact, location and SMS. Some tools are proposed to identify the privacy leakage on Android. However, no one can judge whether the privacy information access is necessary or not. Furthermore, the vast majority of user installed applications contain trustworthiness codes which may leak privacy information.

Smartphone is a source of diverse private information which attracts plenty of attention. For example, the advertising program makes IMEI or other unique device identifiers to track your usage. Cybercriminals can intercept messages, monitor calls, steal personal information, and even listen with the device's microphone if the mobile malwares allows. App Ops was originally introduced by Google in Android 4.3 as a hidden feature. Google removed access to App Ops [1] in Android 4.4.2. However, with root and Xposed modification or custom ROM, app Ops is still possible to be accessed. App Ops lets users block access to parts of sensitive APIs except for the Internet, IMEI and other device identifiers. Privacy Guard 2.0 [2] integrates App Ops. It imposes a notification while running an application that has permissions blocked, supposedly reminding users that it is in operation. Compared to the App Ops and Privacy Guard , XPrivacy [3] can impose alternative restriction which is the sheer breadth and granularity on apps. Although there are some tools, the users have to decide which permissions should be disabled.

Recognizing the risk of privacy leakage, researchers have explored different ways to detect such leakage. TaintDroid [4] is a system-wide dynamic taint tracking and analysis tool, which can track multiple sources of sensitive data. AppsPlayground [5] integrates and reuses TaintDroid. It uses automatic system event triggering and intelligent execution technique to detect privacy leaks and malicious functionality in applications automatically. However dynamic analysis techniques cannot guarantee a complete coverage over all possible conditions to activate app functions of the program. FlowDroid [6] is a static taint analysis tool. Android's lifecycle, context, flow, field and object sensitivity allow FlowDroid more precision to identify privacy leakage than previous Android analyses. William et al. [7] proposed a new static analysis for Android that combines and augments the FlowDroid and Epicc [8] analyses to precisely track both inter-component and intra-component data flow. Although there are plenty of analysis methods to discovery private leakage in Android, a few of them can explain what private leakage is legitimate.

In this paper, we propose a method to solve the above problem. With our analysis results, a normal user can leverage existing fine-grain access control tools to protect their privacy information. Our approach is based on two observations. First, a plenty of malicious operations occur in relatively independent components. Therefore, we propose a method called component partitioning that can separate the app components into primary and non-primary components. The primary components implement the app advertised functionality. Meanwhile, malicious privacy leakage may occur in non-primary components to leak users privacy secretly. Second, some malicious privacy leakages arent triggered by user operation. Since ads or malwares can sneak steal users private information by monitoring system broadcast. In summary, the contributions of this paper are as follows:

- Firstly, we design a component feature based clustering method to divide the components into different clusters. We categorize those clusters into primary components and non-primary components.
- Secondly, we analyze the privacy leakage in applications, and use case based reasoning method to reason malicious privacy leakage from the normal privacy transmitting.

The structure of this paper is as follows. In section II, we present a description of the Android platform and the event triggers mechanism. Section III describes our method of private analysis: First we describe how to partition Android apps components, and then we use case based reasoning method to reasonable privacy leakage. Section IV reports the evaluation results of our approaches. In section V, we review related work in Android privacy analysis and CBR. We finish by concluding in Section VI.

II. Android overview

The Android platform provides the sandboxing mechanism to restrict an app accesses other apps sensitive resource, it also makes permission request as a notice before installing an app. However, many studies show that those defense mechanisms are insufficient to prevent privacy leakage on the Android platform. Almost every app access sensitive resources in Android, its hardly to reject the entire privileges request. In this paper, we focus on how to distinguish high risky privacy leakage from a plenty of normal privacy transmitting.

A. Adversary model

The smartphone is a privacy-intensive device; it contains contacts, photos, SMS and other privacy information. The existing studies show that not only malicious apps, but also normal apps leak privacy information. Ad revenue is the main income for some free or low cost applications. Although there are a plenty of mobile Ads, not all of them comply with the private rules. Aggressive Ads can to steal personal information from users or track their use habits. In additional, Ads revenue attract malicious hackers to replace or embed new ads and malicious code in applications. To maximize revenues, app developers may follow Ad SDK guides to request all of the permissions which the advertising platform declared. Advertising platforms gather various privacy information and analysis this information to efficiently push AD. There are several tools can help users perform a fine-grain permission govern which users can manage some permissions after apps installed. With the help of permissions manages tools, users can block some privacy leakage in Android. Although there is no way to distinguish who is legitimate to access sensitive resource, its hardly to decide which permission request should reject. It is necessary to privacy leakage by static taint analysis.

B. Android permission

Android applications treat each other with distrust principle. The applications are isolated from each other and do not have access to the private data of other applications. The Android employs mandatory access control model which request permissions before an app install to govern the sensitive resources. The four protection levels of permission are normal, dangerous, Signature and SignatureOrSystem. Ordinary applications can request normal and dangerous permissions. Only the manufacturer or system pre-installed applications can use the SignatureOrSystem permissions.

Android developer document recommends that app developers should minimize the number of permissions. To reduce the risk of unintentionally misusing sensitive permissions, do not access to these permissions. Only if permission is required for the app to function, do request it. It is preferable to design an app in a way that does not require any permission. Although a part of app developers want to minimize the number of permissions, they must satisfy the permissions declaration of advertising platforms. Most of free applications embedded one or more advertising modules. In additional, hackers inject malicious codes into applications. Those malicious codes are often encapsulated in background components. We deem the above components as non-primary components; they implement the functions unrelated to the apps advertised functionality.

We list three advertising platforms in Table 1 as example. Advertise platforms lively in various application markets. To max revenue from advertisements, developers usually will choose an advertising platform that has high price of CPC, CPM, CPA, CPS or CPT. Android advertising platform gather some privacy information to accuracy push AD.

C. Android event trigger mechanism

There is no main function in Android application. The essential element of an Android app is component. There are four different types of components, activities, services, content providers and broadcast receivers. Each component can be as an entry point to an Android app. Intent is a messaging object. It can be used to request an action from another app component. Android employs Intent to pass messages between different components. It can be used to start an activity or a

TABLE I. THE PERMISSION OF ADVERTISING PLATFORM

Permission	Advertisement Platform		
	Google Ads	Wooboo	Youmi
INTERNET	●	●	●
ACCESS_NETWORK_SATE	○	●	●
READ_PHONE_STATE		●	●
WRITE_EXTERNAL_STORAGE		●	●
READ_SMS		○	
READ_CONTACTS		○	
ACCESS_WIFI_STATE			●
SYSTEM_ALERT_WINDOW			●
GET_TASK			●
INSTALL_SHORTCUT			○
ACCESS_FINE_LOCATION			○

● Required ○ Optional

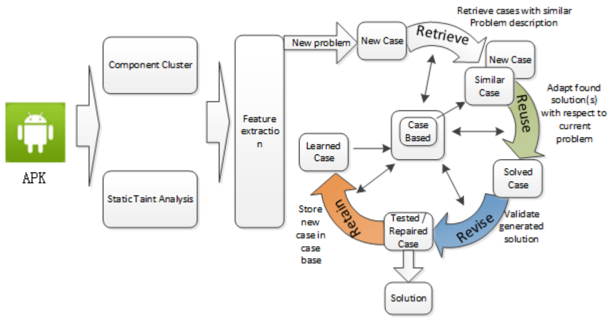

Fig. 1. Case based reason privacy leakage analyze framework.

service, or deliver a broadcast. One or more components can exist as an independent entity and implement an individual task.

As every component can be treat as an independent entity, a developer can add one or more components into an application without modify existing components. This feature introduces a problem that various additional components are injected into Android apps. The ADs request developers add some components into their apps for gathering users information and precise push advertisement. The hackers are also leveraging this feature. Java has been widely used to develop Android applications. There are some tools can decompile and repacked Android app, such as Apktool [9], dex2jar [10] and jdgui [11]. For example, a hacker can download a hot app and inject a broadcast receiver and a service in it. Then, he advertises this repacked app in a market to pretend the hot app. Since smartphone is a privacy-intense device, it employs permissions to restrain app access sensitive resources. However, it cannot restrict a component to access API which is in the range of app permissions. There is no information to explain why a component leak users private to ads, developers or hackers. Therefore, it is necessary to find a method to reason what privacy leakage in one or more components is illegal or legal.

III. METHODS

In this section, we next present our framework to analyzing the privacy leakage that threatens user security. Figure 1 gives an overview of our framework. Our framework consists of four components: static analysis, a component cluster and feature extraction module, and a case based reasoning engine.

There are two static analysis tools : Epicc and FlowDroid. The former can outputs the relation between various components from Android applications. The later performs a static taint analysis and exports tainted paths in all components. The component cluster and feature extraction module divide components into different clusters and extraction a plenty of feature for analyzing. The case based reasoning engine consists of four functions: retrieve, reuse, revise and retain. Soot [12] is a Java optimization framework. It is a basic element of Epicc and FlowDroid. It provides four intermediate representations to analyzing and transforming Java byte code and supports Instrument Android Apps in 2013. The Epicc is a sound static analysis tool on Android platform; it can discovery inter-component communication in Android applications. The FlowDroid is an open source project; it provides a highly precise static taint analysis for Android applications. It treats each component as an independent module. We gather various static features by employing those static tools.

We analyze several applications' Java byte code manually to analyze privacy leakage in Android applications. We found non-primary components often poste threats to users' privacy. Hackers often replace Ad or embed new Ad to hijack developers' advertising revenue. Injecting malicious codes to carry out bad behavior is far less than steal advertising revenue. Some aggressive Ads gather users' information to accurately push advertisements. To protect users privacy, the Ad should not request more permissions except INTERNET permission. Usually the non-primary components implement some functions that are not consistent of the application advertised feature. From above description, we assume that privacy leakage is legitimate in primary components, since it realizes the application declared functions. We propose a method to partition applications' components into the primary and non-primary components. We ignore the privacy leakage in applications' primary components. We introduce a case based reasoning (CBR) method to analyze privacy leakage in non-primary components.

A. Application component partitioning

We observe that non-primary components locate a self-defined directory usually; the locations of primary components are similar to the package name and have an activity as an entry point. In additional, there are strong link between components in primary components. We define a function to evaluate the similarity of component names. It defines as follows:

$$Dis_{Cn_i,Cn_j} = \frac{\sum_{k=1}^{k=m} \mid x_k - y_k \mid}{m} \tag{1}$$

Where Cn_i and Cn_j are two different components names. The name of component split into several strings by dot. For example, the component of com.xxx.yyy.MyServicelabel can be express as a string array (com, xxx, yyy, MyServicelabel). Every component name can be express as a string array. Two components could have different string array, m is the min length in two string array. If x_k equals y_k, the express of $\mid x_k - y_k \mid$ returns 0, otherwise returns 1.

We can get a similarity matrix by computing distance between pairs of components. Later, we get several component clusters based on the similarity matrix. Then we choose a cluster which has a minimum distance with the package name as a candidate primary component cluster, the others as non-primary components clusters. We count the number of times component call between candidate primary component cluster and non-primary components clusters. If the number of time is greater than the threshold, merge two clusters as new candidate primary component cluster. The relation of call between primary cluster and a non-primary cluster define as follow:

$$\begin{aligned} CallRelation = & \sum_{i=1}^{m} iscall(cn_i, NonPcluster) \\ & + \sum_{j=1}^{n} iscall(ncn_j, Pcluster) \end{aligned} \tag{2}$$

Where m is the number of components in primary cluster and n is the number of components non-primary cluster. If $cn_i \in Pcluster$, it has a ICC to non-primary cluster, the function of iscall return 1, otherwise returns 0.

Algorithm 1 Components partition

```
Input:
component list, package name, ICC results.
Output:
Cluster label.
Method:
1:  for $i = 1$; $i \leq componentlistlength$; $i++$ do
2:      for $j = 1$;$j \leq componentlistlength$; $j++$ do
3:          $simMatrix[k][j] = Dis(cn_i, cn_j)$;
4:      end for
5:  end for
6:  $[cluster - label, clusternumber] = clusterbyName(simMatrix)$;
7:  $candidate_primary_cluster = getprimarycluster(componentlist,$
8:  $clusterlabel, packagename)$;
9:  $flag = true$;
10: while $flag$ do
11:     $flag = false$;
12:     for $i = 1$; $i < clusternumber$; $i++$ do
13:         $Crelation[i] = callrelation(Pcluster, NonPclusteri)$;
14:     end for
15:     $[maxrelation, mclabel] = max(Crelation, cluster - label)$;
16:     if $maxrelation > threshold$ then
17:         $merge(Pcluster, cluster - label, maxrelation)$;
18:         $flag = true$;
19:     end if
20: end while
```

B. Case-based reasoning privacy leak classification

An application consists of several components. Android employs Intent to pass message, it is a key element of Android event mechanism. There are several conditions to activate a private leakage in a component, such as a broadcast Intent, a user touch, or a system event. From Figure 2, Pjapps (MD5: 82EDFDC5623AC3182B46748A9517799B SHA1: 0D90CCFAE4BB1AD17DD55768F380406EF3B0ECED CRC32: 8865A1BF) is a malicious app, it gathers serial number of the SIM and subscriber ID, and then send them to remote service. The malicious component is activated by two broadcast receivers. Those broadcast receivers monitor system events and activate malicious codes. Manual analysis malicious private leakage gives the inspiration to us. The context of private leakage could help security expert discovery malicious private leakage. We try to simulation the process of security expert analysis applications. Because of the context of privacy leakage vary from case to case, its methods are a finite set. To resolve this issue, we leverage existing case-based reasoning to accommodate our privacy analysis.

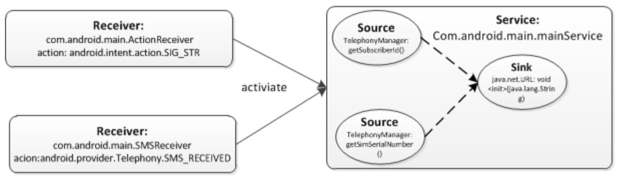
Receiver:
com.android.main.ActionReceiver
action: android.intent.action.SIG_STR
Receiver:
com.android.main.SMSReceiver
acion:android.provider.Telephony.SMS_RECEIVED
activiate
Service:
Com.android.main.mainService
Source
TelephonyManager:
getSubscriberId()
Sink
java.net.URL: void
<init>([java.lang.Strin
g)
Source
TelephonyManager:
getSimSerialNumber()

Fig. 2. Malicious component activate sample.

The study of cognitive science generates CBR. It stimulates a desire to understand how people remember information and are in turn reminded of information, and subsequently recognize that people commonly solve problems by remembering how they solved similar problems in the past [13]. The CBR provides a problem solver; it solves a problem by explicitly trying to reuse a solution from a similar past problem. The CBR engine retrieves cases from a case database and seeks the similarity of cases in the database to the current problem description. In additional, a retrieved case should be suggested to reuse in a CBR system, either with or without revision. Finally, retaining new cases should increase case database of the CBR system. In every case, it has a lot of properties, each given a value, and possibly a weight coefficient denoting the property's importance as part of the whole case. A vector of triplets represents a case. A property is an element of a vector; it contains property name, weight and property's value. A case can be defined as follows.

$$\begin{aligned} C &= \{E_0, \dots, E_k\} \\ E_i &= (P_i, \omega_i x_i) \end{aligned} \tag{3}$$

Where P_i is the property name, ω_i is the weight and x_i is the value assigned to this property.

The similarity measure is used to evaluate difference between two cases. The k-Nearest-neighbor Heuristic is a well-known measure function.

A privacy leakage case contains various properties. A sink point is a function; it associates one or more source function. A privacy leakage case may need multiple permissions. Therefore, some properties are multi-value. We use multi-bits to express those attributes.

CBR process of validation based on the Euclidean norm, every criterion is measured equally. For every CB, both of the numbers of properties in a case and the weighting coefficient of each property are different for all cases in case-library. Similarity measure may be multiplied by a weight factor. However, feature weight is insufficient to show certain preference to a certain criterion. It is hard to express indifference or strict preference towards differences between two cases. We use preference theory functions for similarity measurements between cases in our case-based reasoning model. A preference functions $p_i(d_i)$ allows for each criterion, so each attribute reflects a preference level of two attributes that range from 0 to 1. If the target case and source case are indifferent to each other on the ith attribute, then $p_i(d_i)$ =0. If the target case or source case is strict to each other on the ith attribute, $p_i(d_i) = 1$. Several preference functions were proposed, which cover a large of the practical situations [14].

Retrieve task is a core in a CBR system; it starts with a problem description and end when (a) similarity previous case(s) has been found. The retrieve task consists of four subtasks, identify feature, initially match, search and select. The subtask of identify feature filter out noisy problem descriptors and infer relevant problem features. The initially match retrieves a set of plausible candidates. The task of select is to choose the best match from the set of similar cases. Seeking the best match is a key process in a CBR system. Every case attribute can determine the similarity of the problem (target) case to case in the case-library. To provide a measure of the similarity of the case in the library to the target case, we calculate the sum of the similarity of all attributes. The context of private leakage is a composite attribute; it has one or more elements. We define context information as a sub-concept to the privacy leakage concept. Then we define an algorithm to calculate global similarity

Whether a privacy leakage is legal depends on a best matching previous case. If a target case has several best matching previous cases which have the same similarity, we get the solution by voting. If the majority of that matching cases show that privacy leakages is legal, the target case is legal; otherwise, the target case is illegal.

IV. Experiments and discussion

In this section, we present the results of our experiments in reasoning about privacy leakage. Since the privacy leakage information is a key element of our experiments. We leverage static analysis tools to gather private leakage information. In the following, we describe our experiments data, application component partitioning and case-based reasoning privacy leak classification.

A. Experiment data

For the purpose of the experiments, the static analysis tools, described in the previous sections, were modified and executed to analyze Android Applications. We downloaded 1140 applications from Google play and parsed them. The analysis results fed into the database. To enrich case library, we gather malicious applications from the authors of [15].

Static analysis, often requiring a large amount of resources, may be applied to certain Android applications only. In addition, many developers employ anti-analysis technique to make their applications resistant to static analysis. We chose to analyze Android micro-applications. The static analysis results consist of source functions, sink functions, taint path, and trigger function and so on. Then we manually analyzed the results, and built our case library. We describe some observations of some cases below.

Advertisement SDK introduced one or more components to applications for pushing ad. Almost every free-of-charge app is embedded with one or more advertisement platforms. To precisely push advertisements, some ad components gather and send users private to remote ad servers.

Like for the advertisement cases, some malwares like to pretend hot applications to attract users download. Usually, they repack hot applications and add some malicious components in them. We analysis 9 Android repacked malware families; their malicious code behavior describe as follow.

- ADRD malicious module includes two receivers and one service. The receivers can record system startup and active service component. The service can transfer leak privacy information to remote service.
- BeanBot has two receivers to intercept SMS, monitor system startup and state changing. Whenever those receivers receive an Intent, they start a service to send SMS, leak device information or update application.
- BgSer is a sophisticated Android malware. It defines seven receivers with high priority; for that it can listen several system events, for example system startup, message coming and call, network changing. Those receivers can start a service to carry out malicious behaviors in BgSer.
- Dogwar registers a receiver to receive system startup broadcast. Whenever the receiver discovery system boot completed, it will start a service. The service queries contacts and sends text message.
- Endofay is similar to Dogwar.
- Geinimi has three malicious components. A receiver is used to listen to system startup and start a service. An activity is a fake activity. The service is at the core of malicious module. It can write private information into a file and send users' information by SMS.
- Pjapps defines a receiver with high priority to intercept and send SMS. It also has another receiver to listen the phone signal strength and start a service. The service access internet to transfer privacy information.
- Plankton has only one component in its malicious module. It adds a piece of code in the function of onCreate() which belongs to the main activity. Whenever the repacked application startup, it will invoke onCreate(). Then, it will start a malicious service; the service will transfer privacy information by access internet.
- SndApp is similar to ADRD. Whenever it receive a broadcast with boot completed message, it start a service. The service transfer mobile information to remote service by internet.

There still exist other repacked malware families in our malware dataset, such as DroidKongFu family and its variants. Since the finite capacity of the state of arts of static taint analysis tool, we can't analyze those repacked applications. We select nine repacked malware families to build our malicious cases. There are other malware families in our dataset, for

TABLE II. THE ATTRIBUTES OF A PRIVACY LEAKAGE CASE.

Feature Set	Features
Privacy leak component	Usage of activity
	Usage of service
	Usage of content providers
	Usage of broadcast receivers
Source and sink	Usage of source function
	Usage of sink function
Permission	Usage of permission
Trigger component	Usage of activity
	Usage of service
	Usage of content providers
	Usage of broadcast receivers
Trigger action	Action list
Trigger function	Function list

example Asroot. Asroot use a vulnerability to root Android system. Those applications are beyond the scope of our paper discussion. According to the paper [15], 86% malware are repacked application. In this paper, we focus on the legitimate applications with malicious payloads.

B. Experiments

We completed our experiments, to analysis Android application private leakage. We analyzed the private leakage which occurred in non-primary components. Because the primarrry components associate application major functions, it needs to understand application's semantics in order to analyze private leakage. The non-primary components have limited association with application major functions, so our experiments divide into component partition and reasoning.

1) Experiment 1: The purpose of experiment 1 is to divide application components into primary components and non-primary components. Application component partition is an essential module; it affects both the accuracy and efficiency of our experiment. XMLPullParser [16] is used to parse Android manifest files. From Android manifest file, we get application's package name, component list, permission list, intent-filter list and application name. Epicc [8] is used to analyze inter-component communication in Android application. Based on the static analysis results, we can divide components into primary components and non-primary components by execute our component partition algorithm.

2) Experiment 2: The experiment 2 is aim to reason private leakage. Smartphone is a privacy intensive device; we leverage FlowDroid to analyze Android applications. Comparing with other tools, it supports a context, flow, field and object-sensitivity in static taint analysis. However, it treats every component alone. It's hard to distinguish malicious privacy leakage from normal usage. In order to build our privacy leakage case, we combined inter-component communication information and other context information into the results of static taint analysis to analyze privacy leakage case. Content of a privacy leakage case is defined in Table II.

We use a vector to represent a privacy leakage case. There are four types components in Android, then we use four bit to represent privacy leak component. There are 135 sources and 60 sinks in FlowDroid [17]. Those sources and sinks can be classified into 12 and 14 categories respectively. We can use 26 bits to represent sources and sinks in a case instance. Android platform has more than 130 permissions. We select 26 permissions and use 26 bits to illustrate them. The rest of case attributes are treated in the same way. We leverage the open-source CBR tool and software development kit (SDK) myCBR 3[18] and db4o database to build our prototype system.

C. Experiments results

1) Non-primary component analysis: An Android application consists of one or more components. Different components cooperate with each other to achieve application's functionality. In our experiments, we downloaded 1140 free applications from Google Play. Ad SDK and third-part library are common non-primary components.

The goal of component partition method is to assist in identifying unnecessary privacy leakage in non-primary components. Repacked applications throw a serious threat to smart phone users. It is common that repacks a legitimate application and adds malicious payloads to it. By analyzing our malware dataset, malicious payloads usually belong to the non-primary components. Third-party libraries are widely accepted by Android app developers. Although it is not recommended to use other code, sometimes shared library helps in getting the application quicker to market. Most of the third-party libraries are ad libraries. Third-party libraries have little relationship application advertise function. They also belong to the non-primary components.

In Table III, we list some common non-primary components which are found in our experiments. There are 27 mobile advertising platforms, 4 mobile analytic software. Advertising has emerged as the dominant revenue source for Android developers, especially for free applications developers. Some profit-driven developers added multi-advertises platforms to their applications for revenue. If the developers want to add multi-advertises platform, they may add all of ads platforms' requested permissions to their applications. However, not all of the ads have a fine-grained policy to protect users' privacy. Some studies also show that a plenty of ad platforms have security threats [19], [20], such as abuse permission [21] and leak privacy data [22].

We propose the component partitioning method that help us to locate malicious payloads. We leverage nine repacked malware families to evaluate our method. In Table IV, we list the result of our component partitioning. The result shows that malicious payloads usually create a new package. Our approach success separates nine malicious payloads from nine malicious repacked malware families (189 applications). It is common for hacker repack different applications with the same malicious payload code. Component partition is the first step. It can reduce the privacy leakage analysis work. Since the malicious payloads belong to the non-primary components, we concentrate on analysis privacy leakage in non-primary components.

2) Privacy leakage reasoning: There are a plenty of tools for static code analysis. However, code analyzing is a labor intense working; its' efficiency relies on malicious code analysts' experience. We proposed a case-based reasoning method to assist analysts analyze privacy leakage in Android application. Case library is the basic element of a case-based reasoning system. To create a case library, We manually gather the privacy leakage cases from different applications. Those applications are not only come from Google market, but also

TABLE III. THE LIST OF COMMON NON-PRIMARY COMPONENT MODULE

Non-primary	Type	Module name
Google/ads	Ad	com.google.ads
Domob	Ad	cn.domob.android.ads
AdMob	Ad	com.admob.android.ads
Airpush	Ad	com.airpush.android
Mobfox	Ad	com.mobfox.sdk
Mobclix	Ad	com.mobfox
Tapfortap	Ad	com.tapfortap
Madvertise	Ad	de.madvertise.android
Revmob	Ad	com.revmob.ads
Pontiflex	Ad	com.pontiflex.mobile
Boost beluga	Ad	com.boost.beluga
Tapcontext	Ad	com.tapcontext
Millennialmedia	Ad	com.millennialmedia
Startapp	Ad	com.startapp.android
Mobilead	Ad	com.nbpcorp.mobilead
Inmobi	Ad	com.inmobi.androidsdk
Amazon	Ad	com.amazon.device.ads
Tapjoy	Ad	com.tapjoy
Radiumone	Ad	com.radiumone
Everbadge	Ad	com.everbadge
Sponsorpay	Ad	com.sponsorpay.sdk
W3i	Ad	com.w3i.offerwall
Sponsorpay	Ad	com.sponsorpay.sdk
Everbadge	Ad	com.everbadge.connect
Aarki	Ad	com.aarki
Appflood	Ad	com.appflood
Mopub	Ad	com.mopub.mobileads
Applovin	Analytics	com.applovin
Google.analytics	Analytics	com.google.analytics
Flurry	Analytics	com.flurry.android
Umeng	Analytics	com.umeng.fb.ui
Phonegap	Library	com.phonegap

TABLE IV. THE COMPONENTS PARTITION FOR MALICIOUS REPACKAGE APPS

Repacking familiar	Num	Non-primary component
ADRD	22	com.xxx.yyy.MyBoolService
		com.xxx.yyy.MyAlarmReceiver
		com.xxx.yyy.MyService
BeanBot	8	com.android.providers.update.OperateReceiver
		com.android.providers.update.OperateService
		com.android.providers.sms.SMSService
		com.android.providers.sms.SMSSendService
BgSer	9	com.mms.bg.transaction.SmsReceiver
		com.mms.bg.transaction.PrivilegedSmsReceiver
		com.mms.bg.ui.BootReceiver
		com.mms.bg.ui.AutoSMSRecevier
		com.mms.bg.ui.InternetStatusReceiver
		com.mms.bg.ui.BgService
Dogwar	1	com.dogbite.Rabies
		com.dogbite.Doghouse
Endofay	1	com.YahwehOrNoWay.theword
		com.YahwehOrNoWay.PostingServiceReceiver
		com.YahwehOrNoWay.SMSsmack
Geinimi	69	com.geinimi.AdServiceReceiver
		com.geinimi.custom.GoogleKeyboard
Pjapps	58	com.android.main.MainService
		com.android.main.ActionReceiver
		com.android.main.SmsReceiver
Plankton	11	plankton.device.android.service.AndroidMDKService
SndApp	10	com.and.snd.AirHornSoundService
		com.and.snd.StartAtBootServiceReceiver

TABLE V. THE PRIVACY LEAKAGE REASONING IN NON-PRIMARY COMPONENTS

Privacy leakage reason	precision
Market cases	82.3%
Malware cases	95.65%

come from our malware dataset. Then, we add the analysis results to our case library.

During the manually analysis, we found that different applications share similar behavior to transfer private information. Take Dogwar and Endofay as examples, Both of them have a receiver to listen boot completed event. Whenever the OnReceiver() is called, it will start a malicious service. The service will query database and send SMS. The difference between two malware families is that Endofay defines a high priority to intercept SMS.

When we established case-based library, we take the results of apps' static analysis to evaluate our case-based reasoning system. The non-primary components have a quite difference to each other. Firstly, we take component partition method to process market dataset. We found that 282 applications embedded Google Ads, 2 applications added Boost beluga, 3 applications added Tapcontext and 4 applications added Startapp as their advertising module. Secondly, we take static taint tool to gather privacy leak information. Then, we translate those information into several cases. It hardly judges privacy leakage in non-primary component when the case library hasn't a similar previous case. Most of non-primary components are analytics, crash reporting, authentication, advertisements, cryptography, push messaging and so on. For the state-of-the art static analysis tools restrain, we only discovery part of above advertising platforms have privacy leakage.

Although the static analysis tools can not process obfuscated Java code, native code and Java java reflection, we can get a plenty of privacy leak information from Applications' non-primary components. Advertising platforms takes a great proportion in non-primary components. We also found some gather users' privacy information, such as Tapcontext. Comparing to the other ads platform, Google Ads has strong performance in privacy protect. It just request INTERNET permission. With the static analysis tool, we can't get any privacy leakage by analyzing Google Ads.

With the results of static analysis, we can reason non-primary components' privacy leakage in Android applications. We gather 285 malware cases from malware dataset. Our case-based reasoning system can identify 95.65% malware cases. Since our case library come from malware dataset. To avoid overfitting case-based library, we use 368 cases which are come from market dataset to evaluate our case-based engine. Our engine can distinguish 82.3% case. Comparing with malware cases, its precision has a little low. Since many third party libraries have simlar behavior with malware, for example, they will monitor system event and launch some component on the background. By analyzing market dataset, we find an application who is come from Google market, its' privacy leakage cases has a high similarity with a known repackage family. When we analyze this application manually, we found that the app belongs to "ADRD".

Using Case-based engine to reason Android applications' privacy leakage is dependent on the previous cases. If we can enrich case library, it can be used to distinguish more privacy leakage.

D. Discussion and future work

We propose a method that leverage Android static analysis tools and case-based reason engine to reason apps' privacy leakage. The experiments show that case-based reason can assist analysts analyze Android app. With the cases increasing, the case-based engine will be more and more precise.

Comparing with dynamic analysis, static analysis has better code coverage and more analytical precision. However, static analysis tools also have their drawback, such as state-explosion problem, consuming computer resources and so on. In addition, more and more Android applications developers are employing NDK and encryption to protect their intellectual property. The state-of-the art static analyze tools can't trace privacy leakage in NDK modules.

FlowDroid [6] treats every component as a independent individual, it can't trace leakage between application components. In future, we will try to use other static analysis tools, such as DroidSafe [23] to get more precise privacy leakage information. The case-based reasoning system has an advantage, with the case library increasing; it will have a better classify performance. It also has a fault that the case-based reasoning method relies on previous cases.

V. Relate work

Smartphone platform contains various kinds of applications, such as games, office, e-payment, and social networking. Those characteristics make smartphone becoming a privacy intensive device. Privacy leakage attracts lots of research effort on Android platform. TaintDroid [4] is a system-wide dynamic taint tracking and analysis tool, which can track multiple sources of sensitive data. AppsPlayground [5] integrate and reuse TaintDroid [4]. It uses automatic system event triggering and intelligent execution technique to detect privacy leaks and malicious functionality in applications automatically. But dynamic analysis techniques cannot guarantee a complete coverage over all possible conditions to activate app functions of the program. FlowDroid [6] is a static taint analysis tool. Apps' life cycle, context, flow, field and object sensitivity allows it to be more precise in identifying privacy leakage than previous Android analyses. Klieber et al.[7] proposed a new static analysis for Android that combines and augments the FlowDroid [6] and Epicc [8] analyses to precisely track both inter-component and intra-component data flow. DroidAlarm [24] is a static analysis tool that can identify potential capability leaks and present concrete capability leak paths in Android applications.

WHYPER [25] uses natural language processing (NLP) technology to analyze the application description and identify the need of permissions to the application. VetDroid [26] proposed a dynamic analysis platform that analyzes sensitive behaviors based on perspective of permission use on Android, but it can't cover all of user permissions in the application. Although there are plenty of analysis methods to discovery private leakage in Android, a few of them can explain what private leakage is legitimate. Zhang etc al [17] introduce a bytecode rewriting approach to keep track of private information and detect leakage at runtime. They model the user's decisions with a context-aware policy enforcement mechanism to distinguish legitimate and malicious leaks. Zhou and Jiang [27] systematically study two vulnerabilities in unprotected Android component. They found that a large number of vulnerable apps in popular Android markets with a variety of private data for leaks and manipulation reflect the severity of these two vulnerabilities. AndroidLeaks [28] is a static analysis framework; it seek potential leaks of sensitive information in Android applications automatically.

Case Based reasoning (CBR) is a method, it has been applied in many fields [29]. Watson [30] use four examples to explain CBR is a methodology not technology. H2CBR [31] used six hybrid CBR modules to predict business failure. Slonim et al. [32] propose using fuzzy-valued properties in case representation and design a versatile fuzzy CBR system. CBR has been introduced into security analysis. CBR has been introduced into security analysis. Based on the concept of clone detection and case base reasoning, CBRFD [33] is software flaw detector; it detect security vulnerabilities of codes. ReasONets [34] combines anomaly detection with CBR methodologies to provide situational awareness in case of network incidents. Zhou et al. [35] leverage agglomerative clustering algorithm to analyze program dependency graph (PDG), then group these apps‘ class packages into primary modules and non-primary modules. They detect destructive rider code by comparing the non-primary modules. However, it can't explain the reason of malicious destructive payloads. We combine the aspect of Android static analysis and case based reasoning to reason about privacy leakage.

VI. Conclusion

In this paper,we present a method to analyze unnecessary privacy leakage in mobile applications. Based on the observation that unnecessary privacy leakage may occur in non-primary components, and the non-primary components are loosely coupled with the primary module of the application. We propose a component partition method to effectively locate non-primary components for privacy analysis. Later, we adopt a static taint analysis tool to analyze privacy leakage in the apps. To effectively analyze privacy leakage, we further propose an approach that uses case based reasoning to reason about each privacy leakage. We have implemented a prototype and applied it to analysis apps' privacy leakage in a dataset collected from different markets. Our results show that many non-primary components abuse users' privacy. With precise privacy leakage analysis results, smartphone users could leverage existing fine-grain access control tools to protect their private information from leaking.

VII. Acknowledgments

This work is supported by National Natural Science Foundation of China under grants 61173170, 61300222, 61433006 and U1401258, Innovation Fund of Huazhong University of Science and Technology under grants 2015TS069 and 2013QN120, and Science and Technology Support Program of Hubei Province under grant 2014BCH270.

References

[1] R. Amadeo, "App ops: Android 4.3's hidden app permission manager, control permissions for individual apps!" *Disponibile a http://www. androidpolice. com/2013/07/25/app-ops-android-4-3s-hidden-apppermission-manager-control-permissionsfor-individual-apps*, 2013.

[2] P. Q. Nguyen, "Can we trust cryptographic software? cryptographic flaws in gnu privacy guard v1. 2.3," in *Advances in Cryptology-EUROCRYPT 2004*. Springer, Conference Proceedings, pp. 555–570.

[3] "XPrivacy home page," https://github.com/M66B/XPrivacy, 2015, [Online; accessed 19-July-2015].

[4] W. Enck, P. Gilbert, S. Han, V. Tendulkar, 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," *ACM Transactions on Computer Systems (TOCS)*, vol. 32, no. 2, p. 5, 2014.

[5] V. Rastogi, Y. Chen, and W. Enck, "Appsplayground: automatic security analysis of smartphone applications," in *Proceedings of the third ACM conference on Data and application security and privacy*. ACM, 2013, Conference Proceedings, pp. 209–220.

[6] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel, "Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps," in *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, 2014, Conference Proceedings, p. 29.

[7] W. Klieber, L. Flynn, A. Bhosale, L. Jia, and L. Bauer, "Android taint flow analysis for app sets," in *Proceedings of the 3rd ACM SIGPLAN International Workshop on the State of the Art in Java Program Analysis*. ACM, 2014, Conference Proceedings, pp. 1–6.

[8] D. Octeau, P. McDaniel, S. Jha, A. Bartel, E. Bodden, J. Klein, and Y. Le Traon, "Effective inter-component communication mapping in android with epicc: An essential step towards holistic security analysis," *Effective Inter-Component Communication Mapping in Android with Epicc: An Essential Step Towards Holistic Security Analysis*, 2013.

[9] R. Winsniewski, "Androidcapktool: A tool for reverse engineering android apk files," 2012.

[10] B. Alll and C. Tumbleson, "Dex2jar: Tools to work with android. dex and java. class fil es."

[11] E. Dupuy, "Jd-gui: Yet another fast java decompiler," *URL http://java. decompiler. free. fr/? q= jdgui/accessed March*, 2012.

[12] R. Valle-Rai, P. Co, E. Gagnon, L. Hendren, P. Lam, and V. Sundaresan, "Soot: A java bytecode optimization framework," *Masters Abstracts International, Volume: 41-01, page: 0255.;Adviser: Laurie Hendren.*, 2000.

[13] K. J. Hammond, *Case-based planning: Viewing planning as a memory task*. Elsevier, 2012.

[14] V. Podvezko and A. Podviezko, "Dependence of multi - criteria evaluation result on choice of preference functions and their parameters," *Technological and Economic Development of Economy*, vol. 16, no. 1, pp. 143–158, 2010.

[15] Y. Zhou and X. Jiang, "Dissecting android malware: Characterization and evolution," in *Security and Privacy (SP), 2012 IEEE Symposium on*. IEEE, 2012, Conference Proceedings, pp. 95–109.

[16] A. Slominski, "Xml pull parser," *DOI= http://www. extreme. indiana. edu/xgws*, 2004.

[17] M. Zhang and H. Yin, "Efficient, context-aware privacy leakage confinement for android applications without firmware modding," in *Proceedings of the 9th ACM symposium on Information, computer and communications security*. ACM, 2014, Conference Proceedings, pp. 259–270.

[18] A. Stahl and T. R. Roth-Berghofer, *Rapid prototyping of CBR applications with the open source tool myCBR*. Springer, 2008, pp. 615–629.

[19] M. C. Grace, W. Zhou, X. Jiang, and A.-R. Sadeghi, "Unsafe exposure analysis of mobile in-app advertisements," in *Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks*. ACM, 2012, Conference Proceedings, pp. 101–112.

[20] S. Shekhar, M. Dietz, and D. S. Wallach, "Adsplit: Separating smartphone advertising from applications," in *USENIX Security Symposium*, 2012, Conference Proceedings, pp. 553–567.

[21] M. Ogul, S. Baktir, and E. I. Tatli, "Abused android permissions by advertising networks," in *IT Convergence and Security (ICITCS), 2014 International Conference on*. IEEE, 2014, Conference Proceedings, pp. 1–4.

[22] A. Short and F. Li, "Android smartphone third party advertising library data leak analysis," in *Mobile Ad Hoc and Sensor Systems (MASS), 2014 IEEE 11th International Conference on*. IEEE, 2014, Conference Proceedings, pp. 749–754.

[23] M. I. Gordon, D. Kim, J. Perkins, L. Gilham, N. Nguyen, and M. Rinard, "Information-flow analysis of android applications in droidsafe," 2015.

[24] Y. Zhongyang, Z. Xin, B. Mao, and L. Xie, "Droidalarm: an all-sided static analysis tool for android privilege-escalation malware," in *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*. ACM, 2013, Conference Proceedings, pp. 353–358.

[25] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie, "Whyper: towards automating risk assessment of mobile applications," in *Proceedings of the 22nd USENIX Security Symposium, Washington DC, USA*, 2013, Conference Proceedings, pp. 14–16.

[26] Y. Zhang, M. Yang, B. Xu, Z. Yang, G. Gu, P. Ning, X. S. Wang, and B. Zang, "Vetting undesirable behaviors in android apps with permission use analysis," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, Conference Proceedings, pp. 611–622.

[27] Y. Z. X. Jiang, "Detecting passive content leaks and pollution in android applications," in *Proceedings of the 20th Network and Distributed System Security Symposium (NDSS)*, 2013, Conference Proceedings.

[28] C. Gibler, J. Crussell, J. Erickson, and H. Chen, *AndroidLeaks: automatically detecting potential privacy leaks in android applications on a large scale*. Springer, 2012.

[29] A. Aamodt and E. Plaza, "Case-based reasoning: Foundational issues, methodological variations, and system approaches," *AI communications*, vol. 7, no. 1, pp. 39–59, 1994.

[30] I. Watson, "Case-based reasoning is a methodology not a technology," *Knowledge-based systems*, vol. 12, no. 5, pp. 303–308, 1999.

[31] H. Li and J. Sun, "Business failure prediction using hybrid 2 case-based reasoning (h 2 cbr)," *Computers & Operations Research*, vol. 37, no. 1, pp. 137–151, 2010.

[32] T. Slonim and M. Schneider, "Design issues in fuzzy case-based reasoning," *Fuzzy Sets and Systems*, vol. 117, no. 2, pp. 251–267, 2001.

[33] A. R. Honarvar and A. Sami, "Cbr clone based software flaw detection issues," in *Proceedings of the 7th International Conference on Security of Information and Networks*. ACM, 2014, Conference Proceedings, p. 487.

[34] A. C. Squicciarini, G. Petracca, W. G. Horne, and A. Nath, "Situational awareness through reasoning on network incidents," in *Proceedings of the 4th ACM conference on Data and application security and privacy*. ACM, 2014, Conference Proceedings, pp. 111–122.

[35] W. Zhou, Y. Zhou, M. Grace, X. Jiang, and S. Zou, "Fast, scalable detection of piggybacked mobile applications," in *Proceedings of the third ACM conference on Data and application security and privacy*. ACM, 2013, Conference Proceedings, pp. 185–196.