# Detecting GPS Information Leakage in Android Applications

Siyuan Ma*, Zhushou Tang*, Qiuyu Xiao*, Jiafa Liu*, Tran Triet Duong*, Xiaodong Lin†, Haojin Zhu*
*Department of Computer Science and Engineering, Shanghai Jiao Tong University, China
{Masiyuan, pLL, xiaoqiuyu, 10421381150021, neito, zhu-hj}@sjtu.edu.cn
†University of Ontario Institute of Technology, Canada
†xiaodong.lin@uoit.ca

-

***Abstract*—Location Based Service(LBS) becomes very popular in mobile computing platforms, such as Android. However, it could also leak highly personal information about the phone owner if used by Malwares. It has been witnessed that an increased number of malicious Android applications use LBS to obtain users' locations and transmit them to attackers without users' acknowledgement, causing users' privacy breach.**

**In this paper, we first discuss the common way in which privacy can be breached in Android applications, and then define a classification algorithm for GPS information leakage. Furthermore, we develop a location information leakage detection tool named Brox. Brox is based on dalvik-opcode specification, which uses data flow analysis framework equipped with flow-sensitive, context-sensitive, and inter-procedure techniques to detect potential information leakage path in Android malicious applications. Specifically, Brox uses inter-procedure analysis and dependency calculation to understand the intention for each sensitive operation; by using reachable analysis, connection between privacy access operation and leakage operation is established. More importantly, Brox confirms whether the sending out operation contains location information or not using static taint analysis. At last, we classify the detection results with the help of identification of interaction and non-user interaction entry points in order to discover stealthy leaks of GPS location. The extensive examination results show that the proposed method can effectively detect privacy leakage in Android applications with a high accuracy rate.**

## I. Introduction

In the recent years, we are witnessing the skyrocketing popularity of smart-phones. According to data released by IDC, 207.6 million Android and Apple smartphones were shipped in the fourth quarter of 2012, which is up 70.2 percent from 122 million units in the same period in 2011. Among various computing mobile platforms, Android has the highest market share of any smartphone operating system in the world. This has been demonstrated by IDC reports that, Android had a 70.1 percent share of the market, with iOS at 21 percent in the fourth quarter of 2012. This trend is further propelled with the wide availability of feature-rich Android applications that can be downloaded from Android Market and run on smartphones.

Even though Google manages its own application market, *Google Play*, to get auditing for vendor-provided programs, there exist a huge number of third-party apps in numerous marketplaces. According to Androlib [1], it is estimated that there are more than 650,000 Android Apps until March 1st, 2013 [2]. Different from iOS market policy, in which each application should be reviewed before its publications [3], most Android markets allow the publications of Android Apps without reviewing process. This policy has been criticized since it may make users take the risks of running malicious apps. Further, even apps from Google play are not safe at all [4].

Location privacy is receiving an increasing attention recently due to the popularity of smartphones and ubiquitous location based services (LBS) [5]. According to the report of the Federal Trade Commission (FTC), *Mobile Privacy Disclosures: Building Trust Through Transparency*, real-time geolocation data of mobile users could be used to build detailed profiles of consumer movements over time and in ways not anticipated by consumers. Even though FTC has suggested companies consider offering a Do Not Track (DNT) mechanism for smartphone users, prevention of location data leaking still represents a challenge in practice due to the existence of malicious apps. A good example is a recently discovered malicious app, Secret Tracking [6], which obtains user's location information from GPS and then sends the information via SMS in a secret way. Such an abuse of location data will make mobile user suffer from serious location privacy leaking.

To address this problem, C. Gibler presented AndroidLeaks [7], an automatic tool for detecting potential privacy leaks on Android system. AndroidLeaks successfully applies the existing analysis framework on java to the android applications by translating them into a Java Archive (JAR) file. However, the translation from Android application to jar is not always correct. This incorrect translation could lead to an incorrect analysis result. M. Grace also presented a scalable and accurate zero-day android malware detection framework, RiskRanger, based on filtering applications from their behaviours [8]. But, its results may be inaccurate because it only uses the reachable analysis without taking the taint analysis into consideration. Therefore, more research efforts are necessary toward a more complete and accurate malicious application identification.

Different from other information leaking problems like SMS content leaking, acquiring the location information is a common behaviour for legal applications. Existing research shows that 27 of 50 most widely used advertisement libraries need to acquire users' location information to display advertisement

based on their position [9]. The main challenge of this issue is how to differentiate dangerous behaviours from the legal ones. We analyzed Geinimi [10], a popular malware in android. It is an application that collects users' sensitive data (e.g., location information, phone number and etc), and sends it to the remote server through background services.Based on this analysis, we proposed a classification method to identify malicious-like behaviours (or sinks) by the trigger condition and the privacy leaking sources (the malicious behaviour holding privacy information).

To address this issue, we develop Brox, a static privacy leak detection framework in Android application. Inspired by WALA [11] which provides static analysis capabilities for Java bytecode, JavaScript and related languages, Brox inherits its correctness and effectiveness by using inter-procedure analysis framework. Under such a framework, we can handle the information passed from one procedure to another, then establish the connection between the privacy-getting action and the privacy-sending action. What's more, our framework can distinguish different actions triggered by users with the proposed classification mechanism. Compared with the previous works, our framework is "smarter" on deciding which sensitive code can be executed and thus could give a more accurate report with information flow between the procedures.

Our contributions in this paper are summarized as follows:

1) We present Brox, a static analysis framework aiming at detecting Privacy leaks in Android application. The proposed Brox analysis framework, which uses flow-sensitive, context-sensitive, inter-procedure technique for taint analysis, can produce more accurate results.
2) We introduce a classification mechanism aiming at evaluating the risk of potential privacy leaking action. This classification mechanism can speed up the manual confirmation process during the analysis.

The following sections are organized as follows. Section III introduces the Brox inter-procedure static analysis framework and our identification method to detect the potential risky location information request. Section IV shows the completeness and soundness of Brox and the correctness of our identification method.

## II. Classification of location privacy leaking and attack model

In this section, we give a classification of app which have the location privacy leaking problem, and then we introduce our attack model.

### A. Classification Method

There are many applications which need to get and use the users' GPS information. Among them are many malicious applications within varying degrees of mal-behaviors. By researching the privacy leak behavior, we found that malicious application generally acquire the user's private information at first and then sent it through SMS or network. To have a better understanding of these applications and make accurate judgment on them, we need to classify the malicious application into several types.

| Sending method<br>Request Information | only location information | with extra information |
|---|---|---|
| network | low risk | high risk |
| SMS | high risk | low risk |

TABLE I
Classification Diagram

As table I, we first consider the behavior collecting and sending the user's private information without authorization as privacy leaking behavior. After confirming such behavior of a certain application, we further evaluate these applications in two dimensions: the leaking contents and the triggering condition of malicious behavior.

According to the leaking contents we can divide the privacy leak behavior into two types.

1) The behavior which only leaks the GPS information of the devices.
2) The behavior which leaks not only the GPS information but also other private information such as IMEI ID and phone numbers.

We define the leaks only containing GPS information as low-level-risk behavior and the leaks containing more private information as high-level-risk behavior. GPS information alone may not compromise users much since others do not know who is using the device. But if other private information like SMS message or IMEI ID were leaked out, the attacker can trace the user's identity according to these information. Once user's personal information and GPS information were known by the attacker, users can be easily attacked.

According to the triggering condition of the leaking behavior we divide the privacy leaking behavior into three types.

1) The behavior triggered by user interaction with the application.
2) The behavior triggered automatically by background services.
3) The behavior triggered automatically by background services and user interaction.

The first one is triggering the leaking behavior while the user is interacting with the application. In this case, the application must be at runtime and the user must take some specific actions if the leaking behavior want to be triggered. The second one is triggering the leaking behavior automatically by the background services registered by the malicious application. By using the background services, even if the application is closed, the private information can still be collected. Besides, the background services can collect the updating private information consistently. In this way, even if the user's GPS information changes frequently, the attacker can still get the accurate location information of the user. The third type of leak can be triggered by both background services and user interactions. In this case, the benign application got authorization from the user. It can continually send the

user's location without further permission. After considering the three types of privacy leaking behavior, We think the risk of leak triggered by background services only is superior to others.

### B. privacy leak attack model

We have manually analyzed android malicious applications and noticed that most malicious applications' risky behaviour focus on collecting and sending the users' private information. FakeFlash, an Android Malware in the disguise of an Abode flash player, collect user's phone number and phone's IMEI information and then send these information by posting data to remote server. So a complete path from getting privacy to sending privacy should be included in a complete attack model.

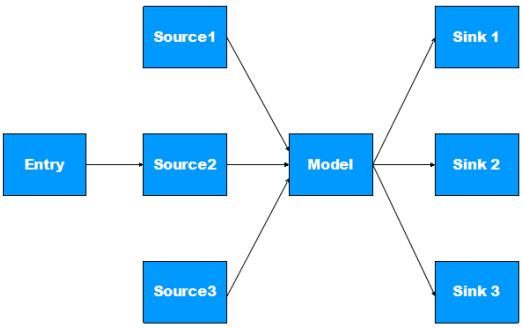


Fig. 1. A abstraction of privacy attack model

Like figure 1, we define the following specific behaviours as the potential privacy leaks of Android application.

1) Collect user's private information using android framework API.
2) Transform private information into another form.
3) Send the transformed information to remote phone or remote server.

In the attack model, we define *Entry* as the triggering action which leads to privacy leak, *Sources* as the action which collects sensitive information, Sinks as *Privacy* sending action. A path connecting a source and a sink is regarded as a *Confirmed path*.

## III. System Design

There are two challenges need to be addressed in the considered problem. The first one is how to build the call graph and get the API parameter. The second one is how to identify the trigger events of a privacy leaking action.

For the first challenge, we use Brox for static analysis. Since the re-compiled code is not reliable, Brox uses dalvik-opcode specification to get more accurate results. Brox is designed based on inter-procedure technique which solves the first challenge.

For the second challenge, in our design, we predefine a series of policies to identify the entry point of Android and then perform taint analysis. We report the existing requesting

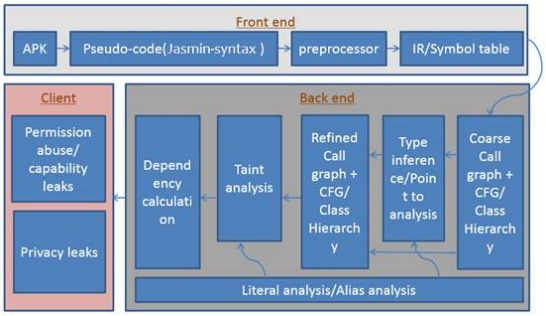


Fig. 2. The Architecture and Workflow of Brox

path of location information with the entry points. With few efforts on manual examination, we can determine whether this location information request is legitimate or not.

Figure 2 shows an overview of our approach. For android applications, Brox extracts it to pseudo-code by using Dedexer. The jasmin-syntax style pseudo-code generated by Dedexer is identical to the bytecode for Android virtual machine. Performing analysis on the android pseudo-code ensures the accuracy of taint analysis.

### A. Building the call graph and getting the parameter

To solve the first challenge,we need to build the call graph and get the parameter. For the call graph, we have two steps. Specifically, the first step is to generate the control flow graph (CFG). The second step is to complete the call graphs based on the CFG.

*1) Call graph Generation:* CFG presents the program execution intra-procedure. In CFG, each node is Three Address Code (TAC) of original instruction and the edge represents the control flow of the instruction. In some cases, we need to take the special steps to transfer the jasmin-style code to TAC. For example, "sparse-switch" is translated to "if-eql" instruction.

```
1
2       new-instance      v0,java/lang/Thread
3       new-instance      v1,ll/ap/ken/LlApKenActivity$1
4       invoke-direct     {v1,v2},ll/ap/ken/LlApKenActivity$1/<init>      ;<
            init>(Lll/ap/ken/LlApKenActivity;)V
5       invoke-direct     {v0,v1},java/lang/Thread/<init>;<init>(Ljava/lang/
            Runnable;)V
6       invoke-virtual    {v0},java/lang/Thread/start      ;start()V
```

Listing 1. A method to start a thread

For inter-procedure, a basic call graph is built on the combination of comments issued by Dedexer and the result of Class hierarchy analysis. To complete the call graph, we apply type inference and const string propagation to get the real class name and method name of the callee. Listing 1 is pseudo-code to start a thread extracted from malware Loozfon [12]. To solve the target of the forked method, type inference is enough. For example, at line 2, type "ll/ap/ken/LlApKenActivity$1" is generated (all generate/kill/assign is used in the transfer function for type inference), and at line 5, we assign the type v1 to v0. At back patching stage where the refined call graph is made, when we get instruction as line 6 presented, a query for the type of v0 is sent to the related element residing in

abstract domain (semi-lattice), and then we get all possible types (classes) of the callee for this point.

*2) Getting the parameter:* We use const string propagation to get the parameter of the sensitive operation. For example, *getLastKnownLocation* is the location related API which returns the location information from GPS when called with parameter *LocationManager.GPS_PROVIDER*. Similar to type inference, we define *LocationManager.GPS_PROVIDER* as const string, and transfer function for const string propagation logs the generate/kill/assignment of the const string. And when Brox encounters *getLastKnownLocation* related operations at dalvik-opcode level, we query the string information in abstract domain for const string propagation and thus obtain the real intent of the API.

### B. Identifying the triggered events

To deal with the second challenge, Brox extracts the classes and methods defined by the authors and recognizes the entry point based on its super class and method name. For example, when the pre-processor encounters a method whose super class is *Android.app.Activity* and method name is *onCreate*, it will be marked as an entry of the function. Because when the android framework starts the application, this method will be called to initialize the component of this activity. We consider the overloaded method of Activity, Service, BroadcastReceiver and Listener as the entry point of an application.

The results of the analysis can show the dependency among the variables. To confirm certain parameter in API which holds the sensitive information, we proceed backward slicing until we hit a sensitive operation. The result of backward slicing is also a representation of the taint flow graph. Therefore, Brox has the ability to analyze the transformation and propagation of the privacy information. It traverses the CFG to search the sending action(sink) by exploiting the deep-first search algorithm. Then Brox tests whether the action is reachable from certain entry points. After that, Brox builds a dependency graph for each parameter related to the sending action. With the help of built-in API summary, Brox proceeds backward slicing. Then we check the leaf of the slicing results. By checking the information of the dependency, Brox reports a confirmed path if it finds that the data is obtained from the source.

Brox has the ability to detect the privacy leaking behaviours in android applications. However, legal applications may also request the location information such as Google map. To distinguish these legitimate applications from the malicious ones, we propose a classification mechanism.

*1) Information Sources:* There are several different ways to retrieve information through Android API. Therefore, there are different types of source information.

1) Location information. The LocationManager in Android provides a series of methods to deal with location information. Once an application got a LocationManager ojbect, the application can handle the location information by one of the following three methods. First, it can query the list of all LocationProviders for the last known user location. The second approach is that it could register/unregister for periodic updates of the user's current location from a location provider. The last approach is to register/unregister for a given Intent to be fired if the device comes within a given proximity (specified by radius in meters) of a given lat/long.
2) Device information. Each smartphone has some unique identifiers, such as IMEI , which is unique for each GSM based on mobile phone and IMSI, which is unique for each subscriber of GSM services. Through TelephonyManager, we can obtain all these identifiers. The class TelephonyManager mainly provides access to information about telephony services for the device. Applications can use the methods in this class to determine telephony services and states.
3) Contact information. Android smart phones are used for personal information management and therefore storing addresses and phone numbers. In the Android SDK, the phone numbers can be accessed by TelephonyManager.

### C. Information leak Sinks

Once the private information is stored in some variables of an Android application, there are several ways to send them to the outside world. Below are some sinks we consider in our work.

1) SMS. Application can send information to other phones or organizations by using the **smsManager** object. Method **getDefault()** can pack the private information in the message. The message can be sent to the destination by the sendTextMessage() and the sendMultipartTextMessage() method.
2) Network. The application can also access the internet through the class socket. The method Socket(String dstName, int dstPort) creates a new streaming socket which is connected to the target host after setting the parameters dstName and dstPort. The dstName is the IP address of the server and the dstPort is the port number of the server.

Using attack model discussed above, we specify the *entry*, *source*, *sink* and their relationship in the configuration file. According to the specification, Brox can address the specific problem and get the corresponding results.

## IV. Experiment and Evaluation

To verify effectiveness and efficiency of Brox, we examine our method by comparing the results of Brox with the existing approaches to GPS information leakage analysis. By analyzing the action mode of Android applications, we examine the classification method described in the previous section. Our experiment is shown in Fig. 3, the analyzer reads the suspicious application, and runs Brox on each entry point. It then analyzes the results and classifies them. The experiment was implemented on a Personal Computer with Core i5 Processor and 2G RAM running Windows 8 professional Operating System. We collect the Malware samples from contagio mobile

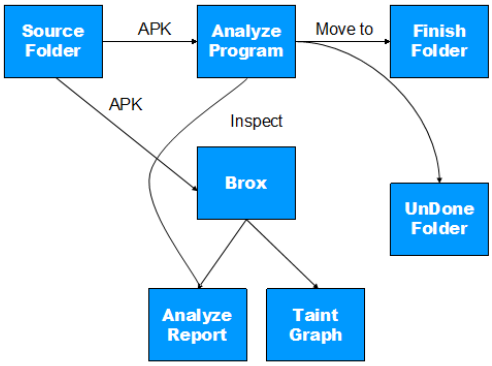


Fig. 3. the overview of experiment platform

| Application Name | Found Possible Privacy Leak | Found Confirmed Privacy Leak |
|---|---|---|
| Plankton | | ○ |
| Geinimi | | ○ |
| Mobile Spy | | ○ |
| MobileMonitor | ○ | |
| spyera | ○ | |
| MobileTX | ○ | |
| FakeFlash | | ○ |
| Spybubble | ○ | |
| iits_sypoo | ○ | |
| Boxer | | |

TABLE II
ANALYZE REPORT

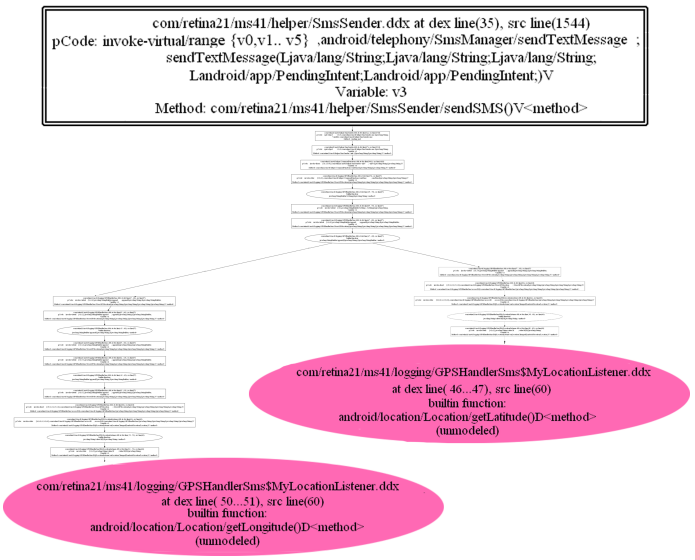


Fig. 4. The Analysis Result of Mobile Spy Generated by Brox

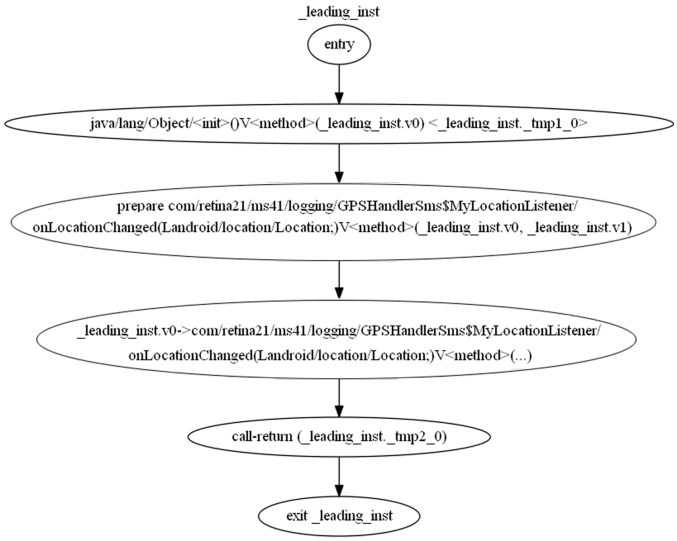


Fig. 5. The Entry Point Graph Generated by Brox

blog [13]. Additionally, we also collect some monitor applications in the report [14]. All test cases and test results can be found at our website [15].

TABLE II shows that Brox can detect most of malicious actions of Android applications, and is able to generate an elaborate report of privacy leaks in Android applications. Let us take the example of geinimi, a malicious application that collects user's information in the background and sends them via sms. Brox is able to detect most of the geinimi's variants collected from contagio mobile [10]. Moreover, by using apktool, a repacking tool for Android, we further migrate GeiNiMi virus into a clean Android application and make some obscurity on the library. Our experiments show that Brox can still detect location information leakage by the geinimi virus. Figure 4 is the analysis results automatically generated by Brox for Mobile Spy. The node surrounded by two lines stands for the sending action. The picture shows that the application sends the privacy information via *sendTextMessage* API. The *variable* label represents data dependency. For example, the first node with label "v1" indicates that the sending information is stored in the "v1" register, and the edge linked to "v1" stands for that the "v1" register depends on the parameter of "sendSMS". Thus, we can determine the data dependency among variables. The node colored with pink is the source of privacy leak. Obviously, a confirmed path exits in this application. Therefore, we can conclude that the malware acquires the user's location information by using *getLatitude* and *getLongitude* API. The entry point of this privacy leak is shown in figure 5. The source of this malicious application is in *LocationListener*, and a listener notifies the program when it detects the change of location. This malicious behaviour is triggered without user's interaction. Using the information above, we know that Brox detects a privacy leak since the malware sends user's location information through SMS. According to the previous definition, this privacy leak can be

| Application Name | Entry Point | Collect Information |
|---|---|---|
| GeiNiMi | onLocationChanged (registered in background thread) | User's IMEI code and Location |
| MobileMonitor | onLocationChanged (registered in a Broadcast Recevier) | User's IMSE code and Location |
| SpyEra | onCreate (from a background service) | User's Phone Device |
| Spybubble | onLoactionChanged (registered in a background service) | User' s phone number, IMSE code and Location |

TABLE III
THE BEHAVIOUR OF SOME MALICIOUS APPLICATION

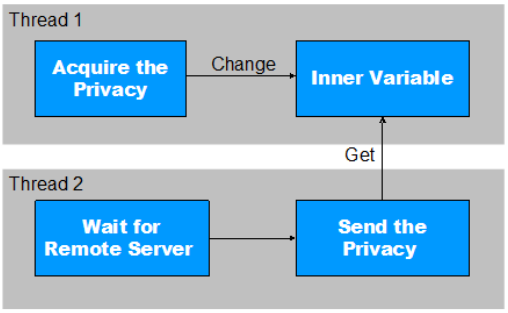


Fig. 6. The Concurrency Problem

considered as a risky behaviour. After obtaining the analysis result, we further apply our classification method proposed in Section II-A. We collect the confirmed privacy paths of two applications, Geinimi and MobileMonitor, and then analyze the results. Brox also reports unconfirmed privacy leak paths of SpyEra and SpyBubble. We then manually analyze these applications to determine what type of sensitive information (or privacy) they collect and how they collect them. From TABLE III, we can know that most location information leaks are related to the function *onLocationChanged*. According to the Android Document [16], the *onLocationChanged* method is called when the user gets a new location. Moreover, the *LocationListener*, the class of *onLocationChanged* method is registered in a background thread or service. We can conclude that the sending behaviour is completed on the background without user's interaction. Regarding the information collected by the application, most malicious applications collect the user's IMEI and IMSI code. Moreover, Spybubble also collects the user's phone number which is generally considered as a more secret information. The general characteristic of these information is that the attacker is able to identify the phone's owner. By combining the location information with identity information, the malicious application can trace the precise position of any person. our experiments show that the proposed classification mechanism can successfully identify the risky behaviour of malicious applications.

### A. Concurrency Problem

According to TABLE II, we found that Brox framework reports 9 possible privacy leak out of 10 samples, however, Brox only report 4 confirmed privacy leak out of them. After manually inspecting these programs, the privacy leak path can be demonstrated in Fig. 6. Let us take an example of Spybubble, a monitor Malware. It gathers user's location information in the background using the *getLongitude* and *getLatitude* API. It then packages the position information into a XML-liked string and stores the string into a global variable. Another method sends this variable to the internet as soon as the remote server is active and can be reached. By using this method, the Malware can handle the situation that the user's phone cannot access the internet. This new type of privacy leak action bring a challenge to our analysis framework since static analysis can not simulate the time sequence of an application. Since the sending method and the collecting method belong to different threads, our framework will be confused by this. The proposed solution cannot figure out which thread changes the variable and let this variable contain user's privacy information. To analyze this type of application, we have to identify the concurrency of each entry point and detect the semaphore and mutex between two threads. A method of performing the inter-procedure dataflow analysis on the concurrency application is proposed by the Amir Kamil and Katherine Yelick. It is an efficient analysis technique to identify the race condition and the concurrent parts inside a program. However, in Android, this method is too slow to apply it on our framework. Like QQ, a popular IM program in China, has 1556 entry points and each entry point can be considered as a thread. If we identify the concurrent problem between each pair of entry position, we have to analyze nearly 1.2 million pairs, which will cost too much time and make it impractical in Android. We are trying to find a new algorithm that analyzes the concurrency problem in large scale.

## V. FUTURE WORK

In the section IV, we proposed the requirement of a new method to analyze the concurrency problem specially on android analysis. The experiment result shows that the android malware developers prefer to write the malicious application with more complex data structure model and more complicated logic. That requires the analyzer increase the ability that handle such method. According to a recent report, the android malware is using more techniques to avoid the detection, include dynamic code loading, encryption and native code exploit. The RiskRanker[8] considered these behaviour as a dangerous behaviour. But we can using the dataflow analysis to analyze the dynamic jar just as the code inside the android application. The progress of this aspect results in a more intelligent and accurate analysis framework that can handle the most complicated malware.

Also, considering the table III, we can figure out that the location privacy issue is connected with entry point of *onLocationChanged*. Inspired by this phenomenon, we can derive a probabilistic model of android privacy leaks path. Let $P(M = m, S = s, L = l)$ be the possibility of a method $m$

with super class $s$ have the privacy leak $l$. Using this model, we can efficiently decide the entry point of the privacy leaks inside the android application. For example, if the analyzer knows that *onLocationChanged* is the main entry of location privacy leak, the analyzer can become faster in that it only detects a few subset of entry point and avoid omitting potential privacy leaks.

## VI. Related Work and Conclusion

As a hotspot of analysis in security, the detection and classification of android malware arouse many researcher's concern. Grace proposed a classification system called RiskRanker to evaluate the dangerousness of application based on the behaviour footprint of application. Grace also analyze the advertisements of android application, they find that most android advertisement requires more permission than excepted. This behaviour could lead to person's privacy information leaks. Yajin Zhou, proposed an analysis between official markets and unofficial markets, they find that unofficial market have more malware than the official one and proposed a need for detecting the malicious application.

Related to our future work, Amir Kamil proposed an efficient algorithm using the dataflow analysis to solve the concurrency problem which we stated in section IV. Also, Peng [17] proposed a method to classify the malicious android application. We want to apply this method on the choice of android entry point.

In this paper, we have presented a inter-procedure dataflow analysis framework to analyze the location privacy leaks inside the android application. And for the special attribute of location privacy leaks, we proposed a classification mechanism based on the trigger event and the information collected by the application. During the experiment period, our framework can successfully detect most of privacy leaks inside the program. However, the concurrency of application become an obstacle of our analysis. In future, we will propose two methods to improve the efficiency and accuracy of the analysis.

## VII. Acknowledgement

This research is supported by National Natural Science Foundation of China (Grant No. 61003218 and No. 61033014), Doctoral Fund of Ministry of Education of China (Grant No.20100073120065), and Opening Project of Key Lab of Information Network Security of Ministry of Public Security (The Third Research Institute of Ministry of Public Security).

## References

[1] Androlib, "Accumulated number of application and games in the android market," http://www.androlib.com/appstats.aspx.

[2] "400,000 apps now available in android market," 2 2012, http://www.theverge.com/2012/1/4/2681360/android-market-400000-app-available.

[3] Butler, "Android: Changing the mobile landscape," *Pervasive Computing, IEEE*, 3 2011.

[4] "Malware from google play," http://news.cnet.com/8301-1009_3-57470729-83/malware-went-undiscovered-for-weeks-on-google-play/.

[5] LBS_report, "Three-quarters of smartphone owners use location-based services," PewInternet, Tech. Rep., 5 2012, http://pewinternet.org/Reports/2012/Location-based-services.aspx.

[6] Tecent, "Tecent found a malicious application call "secret tracking"," 2 2012, http://news.zol.com.cn/276/2764269.html.

[7] C. Gibler, J. Crussell, J. Erickson, and H. Chen, "Androidleaks: automatically detecting potential privacy leaks in android applications on a large scale," *Trust and Trustworthy Computing*, pp. 291–307, 2012.

[8] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, "Riskranker: scalable and accurate zero-day android malware detection," in *Proceedings of the 10th international conference on Mobile systems, applications, and services*. ACM, 2012, pp. 281–294.

[9] 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, pp. 101–112.

[10] f secure, "Mobile threat report q4 2012," 2012, http://www.f-secure.com/static/doc/labs_global/Research/Mobile Threat Report Q4 02012.pdf.

[11] T. Watson, "T.j. watson libraries for analysis," http://wala.sourceforge.net/wiki/index.php/Main_Page.

[12] virustotal, "Loozfon info," https://www.virustotal.com/en/file/ec0e0d25aa1de4f38894fb1999d6f21535610ffba15423a02ec993fea1561c66/analysis/.

[13] contagio Mobile, "contagio mobile," contagiominidump.blogspot.com.

[14] f secure, "Mobile threat report q2 2012," 2012.

[15] M. A. I. Team, "Brox," http://www.mobile-app-insight.org/Publications/Publications.htm.

[16] Google, "Android document," http://developer.android.com.

[17] 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 *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2012, pp. 241–252.