A Proposal for the Privacy Leakage Verification Tool for Android Application Developers

Shinichi Matsumoto
Institute of Systems, Information Technologies and Nanotechnologies (ISIT)
+81-92-852-3454
smatsumoto@isit.or.jp

Kouichi Sakurai *
Dept. of Computer Science and Communication Engineering, Kyushu University
Institute of Systems, Information Technologies and Nanotechnologies (ISIT)
+81-92-852-3454
sakurai@inf.kyushu-u.ac.jp

ABSTRACT

Nowadays, smartphone market has been growing rapidly, and smartphone has become essential as a business tool. One of the crucial advantages of a smartphone is an installable third-party application. Number of these has continued to grow explosively.

However, vulnerabilities in smartphone applications are seemed as serious problem. This is not only for the smartphone users, also for smartphone application developers and/or vendors.

Until now, most vulnerability tests on smartphone applications are targeted that has been packaged as a commercial product and distributed in application marketplaces. These tests are performed on dynamically on application binaries.

In this paper, we aim to develop the static vulnerability verification tool that can be utilized for smartphone application developers and/or vendors in the implementation and/or test phase of development process.

This tool intakes source codes and determine where to read the privacy information in the source codes, and determine where to write/send the information in there.

Then analyze the privacy information transfer and/or transform flow and report the possibilities of privacy information leakage to application developers.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging—*Code inspections and walk-throughs*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms

Security, Verification

Keywords

smartphone, Android, static analysis, information flow, covert channel, software testing

*The second author is partly supported by a Grants-in-Aid for Scientific Research (B)(23300027), Japan Society for the Promotion of Science (JSPS)

ICUIMC(IMCOM)'13,January 17-19, 2013, Kota Kinabalu, Malaysia.

1. INTRODUCTION

1.1 Background

Recently, smartphones are getting popular and essential tools for contemporary life, and these are attracting attention to revolutionary tool for business transform. There are a variety of applications suitable to business objectives. Android OS is employed as basis for a large amount of smartphone OS. It captures a huge market share of mobile platform.

As an intrinsic problem of mobile platform, mobile terminals have many privacy information, e.g. owner's names, phone numbers and mail addresses, and so on. In addition, many devices, e.g. GPS and camera work as sensors to take in critical information from the point of view of privacy.

Leakage of these privacy information may result in significant damage to the users use smartphone, and could cause significant damage to vendors who have developed such applications also.

In order to mitigate this problem applications for smartphone should be restricted to access information within smartphones. When the moment user install application to a smartphone, permissions required by the application are displayed and ask to proceed the installation process or not. If user suspect to these permissions are not necessary to fulfill its function, he/she can veto its process. Although in actually, it is widely known there are many malicious application that causes privacy leaks[3]. There are many applications are not declared properly[8]

1.2 Motivation

There are many applications that cause private information leakage built with malicious intent. On the other hand, there are some applications have vulnerabilities unintentionally. These vulnerabilities are built in its design or coding process unintentionally and these incidents also gave significant financial and/or reputational damage to the application

vendor .

To prevent such incidents, runtime permission management scheme built in Android OS is not sufficient. Therefore, application developers have to deal carefully with the privacy information.

For this purpose, we propose the tool for Android application developers to guarantee the security of Android application product. To analyze an application about the harmlessness on privacy, privacy information move/transform flow analysis is necessary. Privacy information flow analysis is based on privacy sink(privacy information leakage point within source codes)/source(privacy information retrieve point within source codes) properties analysis and flow analysis.

1.3 Related works/researches

There are many services and researches on dynamic security verification schemes for Android application.

Google Play [19] marketplace offers Bouncer [10] that is the service provided by Google officially. It has been reported to have a certain effect to reduce malware applications. On the other hand, it is reported the way to dissect the Bouncer system [14].

TaintDroid [7] and AppFence[11] are researches on the technology that dynamically tracks the private information dynamically. TaintDroid monitors the communication between applications in real-time. It determines whether the communication contents are tainted (including private information) and alerts to the user. For the real-time analysis, AppFence uses two techniques. First, it substitutes shadow data contains private information to be protected. Second, it intercepts the network system call and filters data to prevent exfiltration of private information. Both of them have realized the dynamic analysis by modifying the Android kernel.

Application vulnerabilities that cause information leakage is related to the attacks by covert channel. Covert channel is attracting attention as a subject of research theme[4]. In this attack, two or more applications collude to communicate private information, and obtain the permissions other than requested in manifest. Covert channels are also contemplated that communicated between applications directly[13], or via the Android kernel [15].

In addition, there are many researches on information flow analysis method for compiler design and it is utilized widely in industry[1] [2]. These information flow analysis techniques are used for code optimization, and some researches treat that annotation code included in binary to verification.

In this research, private information flow is analyzed statically and much complex than in its analysis for code optimization. In the code optimization, information flow focuses on data value itself.

However, we have to treat information flow focuses on data semantics. For example, in the case a variable "name" contains smartphone owner's name and a variable "address" contains his address, the statement "name += address" kills his name value as the view from the data value. From the view of data semantics, information on name and address are not killed there and possibility to leak is remains same. We propose a tool for developers, a method to keep track of semantically the data beyond the data conversions.

Furthermore, in terms of the tool that is provided to the developers, MCTF (Model-Based Conformance Testing Framework)[12] tests Android ICC (Inter Component Communication) modules in development stage dynamically.

Table 1: Application Analysis Methodologies

	Target Stakeholder		
	Developer	Operator of market	User
Static	Our Proposal	–	–
Dynamic	MCTF[12]	Google Bouncer[10]	TaintDroid[7], AppFence[11]

In summary, the researches are classified from the perspectives.

- Whether the application analysis is performed dynamically or statically, and
- Which application market stakeholders to be offered this technology.

The summary of combination of these classifications is described as Table 1.

1.4 Challenging issues

To accomplish to detect the private information leakage from smartphone applications, there are some subjects to realize.

- Identify and locate private information reading method and its properties within the source codes.
- Identify and locate private information writing/sending method and its properties within the source codes.
- Private information move/transform flow analysis.

In order to complete as a development tool, it is required to integrate these subjects and must be included in the Android development environment[16].

1.5 Our contributions

In this research, we aim to offer the tool that analyze private information flow statically for Android application developers. It is important that inspect before shipment whether the application has a vulnerability. Keeping track the private information flow that could leak by code review takes a lot of effort, and prone to overlook. Therefore, we propose a testing tool to automate this tasks.

For this purpose, we present a method for checking the source of private information within the source codes. In this step, it is clarified what variable is assigned the private information. In addition, we present for checking the leakage point of private information. In this step, it is clarified where the private information is exfiltrated. Next, we intend to track the flow of private information on the source codes.

It prevent to expose to the android application vulnerabilities for smartphone users. This tool can be utilized in the following situation.

1. It can be used as a tool in the usual test and review process. Especially before shipment inspection.
2. It can be utilized in acceptance testing, to verify the code developed another companies/groups.

This should also make significant contributions to the application developers and/or vendors eventually.

2. ANDROID ARCHITECTURE

2.1 Architecture abstract

Linux kernel is the foundation of Android platform.

Android is composed of a variety of device drivers, the framework for mobile communication terminal in addition to the Linux kernel as Figure 1.

Almost Android applications run on Dalvik virtual machine, individually. These applications are described by bytecode run on Dalvik VM. The Dalvik VM interprets these bytecodes and executes them.

Android applications are developed with Java language. Android SDK has a compiler that translates Java source codes to Dalvik bytecode. Android applications are distributed with "apk" format. This format is consisted of binary files (described by Dalvik bytecode), application resources, and manifest file. Manifest file describes the permissions required by application for work functionally.

2.2 Permission System

Permission is a essential mechanism to achieve android application runtime safety[17][6]. It controls access to resources and/or devices within the smartphone device.

When user initiate the application install process, Android system indicates permission details to user. Then user is required to approve to proceed install process. At the time, user has to recognize these permission requests and verdict whether install or not. User must read the description of permissions displayed on the terminal screen thoroughly, and determine these permissions are necessary to complete its function or not.

The application approved to install obtains all of permissions described in its manifest file. Permissions in the Android system are classified according to the four protection levels. Protection levels defined are "normal", "dangerous", "signature", "signatureOrSystem". Detailed meaning of these are described in [22]. Usually, third-party applications can declare only former two levels. Typical permissions defined in Android system are described in Table 2[20].

In the lifecycle of Android application, permissions decided at the install time is immutable. After completion of the install process, Android system cannot divest of permissions of the application.

2.3 Sandbox

Applications run on a Android machine are isolated by the "sandbox" mechanism as Figure 2. The concept of Sandbox is based on Linux process model. Basically, each Android application runs on a dedicated Dalvik Virtual Machine(VM). The Dalvik VM interprets application bytecode. Furthermore, Dalvik VM is executed within the own process. At the point of view, each applications are executed within own sandbox.

Each applications cannot communicate across the sandbox border directly just like Linux processes. These communications are restricted by Android system API.

2.4 Inter-application communication

Android system has an API for inter-application communication which is called "intent". Intent is consisted of some elements as below.

action Identifies the request to process to intent receiver. Android system defines some actions as default. For example, "ACTION_MAIN" requests to start an application, and "ACTION_CALL" requests to make a phone call.

data Specifies the location of information object to be processed in URI form. It is composed of scheme, host, port and path.

category Designates the detailed processing method. For example, "CATEGORY_LAUNCHER" means it should initialize the task, or it should be executable as launcher.

type Specifies a type of data object of intent in the form of MIME type. For example, "text/*" specifies characteristic data type.

component Intent has component element that is delivered to the designated component. Android system delivers intent to the component and instantiates it. Component is classified to "Activity", "Service", "Broadcast Receiver" and "Content Provider".

extra It can contain additional information to pass to the component.

Intent is classified into two categories depend on whether receiver is explicitly addressed or not, former one is called "Explicit Intent" and later one is called "Implicit Intent". Explicit intent has component element and this element indicates the destination component of intent. Implicit Intent receiver declares intent filter, indicates what action to process. However, intent filter does not guarantee receiver process the intent according to the specified action.

2.5 Problems in Android security system

About Android permission system, permissions required by application are indicated to the user when its application installation is initiated and prompted to decide install or not.

However, it is difficult to decide application must be install or not when in installed process. For example, INTERNET permission indicates application requires to obtain the Internet access. According to this description, knowledge on destination server address and/or port number are not described. These permission description are too coarse to decided by user.

3. ASPECT OF ANDROID SECURITY VULNERABILITIES

As described in former chapter, security architecture in Android is based on an isolation with sandbox and access control with permission. However, it is not sufficient to guaranty the security. Some of vulnerabilities on Android applications are reported as follows.

1. Inadequate handling of IMEI number
   In this case, there is a vulnerability in a game application. IMEI (International Mobile Equipment Identifier) number is written in external memory card. To read IMEI number, application has to obtain the READ_PHONE_STATE permission. Applications without the permission can read IMEI number via memory card.

2. Implicit Intent in communication application
   It is a vulnerability in a communication application.

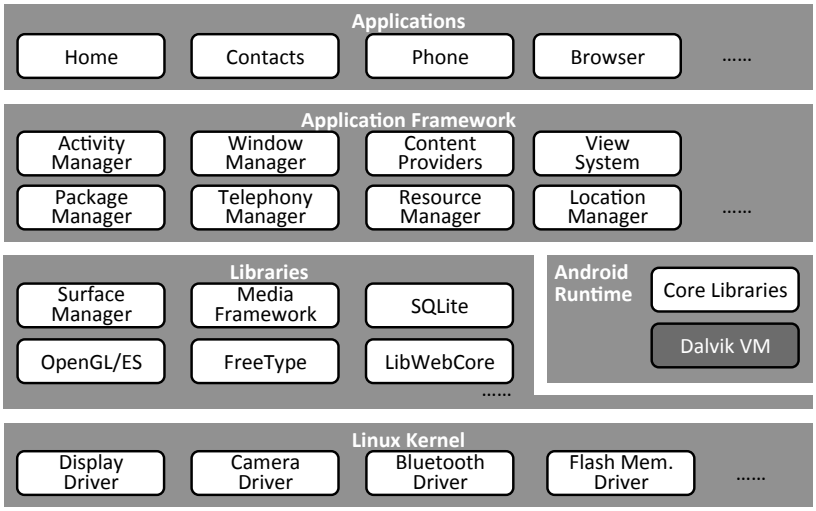

Figure 1: Android Architecture

Table 2: Excerpt of the Android System Permissions

Permission	Protection Level	Description
INTERNET	dangerous	Allows applications to open network sockets.
ACCESS_NETWORK_STATE	normal	Allows applications to access information about networks.
ACCESS_WIFI_STATE	normal	Allows applications to access information about Wi-Fi networks.
READ_PHONE_STATE	dangerous	Allows read only access to phone state.
CAMERA	dangerous	Required to be able to access the camera device.
WRITE_EXTERNAL_STORAGE	dangerous	Allows an application to write to external storage.
ACCESS_FINE_LOCATION	dangerous	Allows an application to access fine (e.g., GPS) location.
ACCESS_COARSE_LOCATION	dangerous	Allows an application to access coarse (e.g., Cell-ID, Wi-Fi) location.

This application throws messages in implicit Intent. Because of this vulnerability, application has no permission can receive these messages.

3. Inadequate management of preference
   In this case, there is a vulnerability in a communication application. It put its own preferences in a directory there did not comply a Android guidance. This vulnerability causes another application that has no permission can read these preferences.

4. TOOL FOR VULNERABILITY TEST

Vulnerabilities described in previous chapter are not elaborated with malicious intention, just blamed for the developers carelessness or lack of skill. These vulnerabilities are elaborated in design or implementation process. It is important to detect and remove before application release.

To achieve it, is useful to make the references for security checkpoint and collate with it when writing or reviewing the code. However, there is a limitation to full check of flaws on human work. In this paper, we describe the vulnerability check tool to utilize at implementation or test stage in development process. Feature of this tool is as follows:

1. Static verification is performed on source code and resource file.
2. Identify the processing flows of critical information on privacy/security and alert them to developers.
3. Implement as an Eclipse plug-in and integrate to Android development environment.
4. Classification based on the severity of critical information.
5. Analysis and alert on the incomplete information flow. It can be used on the spiral development process.
6. To keep up with rapid enhancement of Android API, architecture is developed as configurable and/or expandable.

4.1 Schematics for vulnerability testing

Schematics of the tool we propose as Figure 1. This tool is implemented to integrate to Android SDK[16]. It read

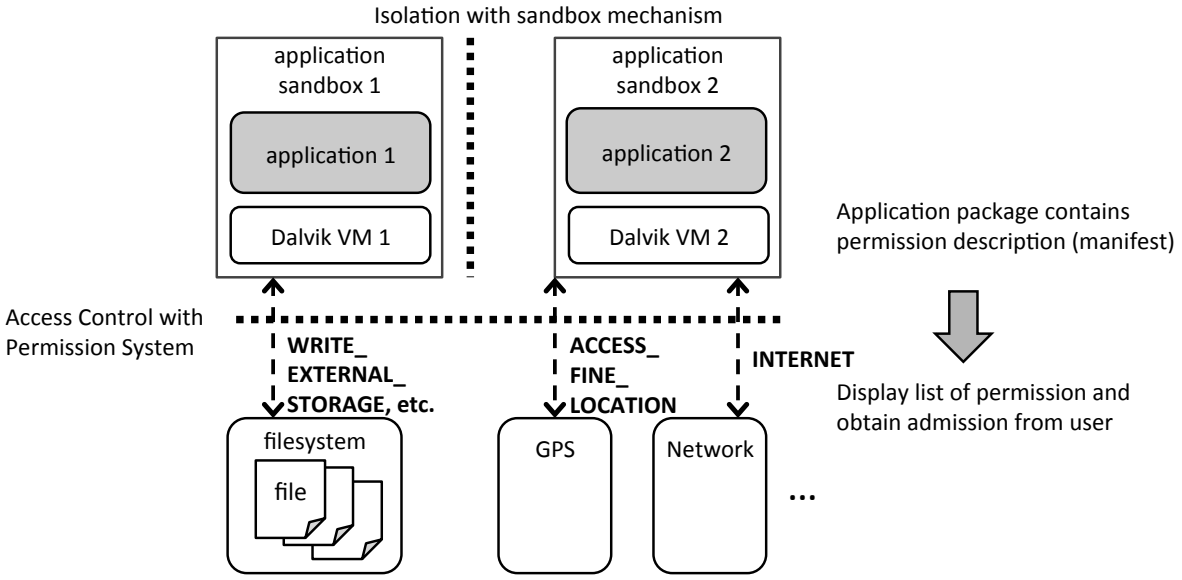

Figure 2: Android Security Model

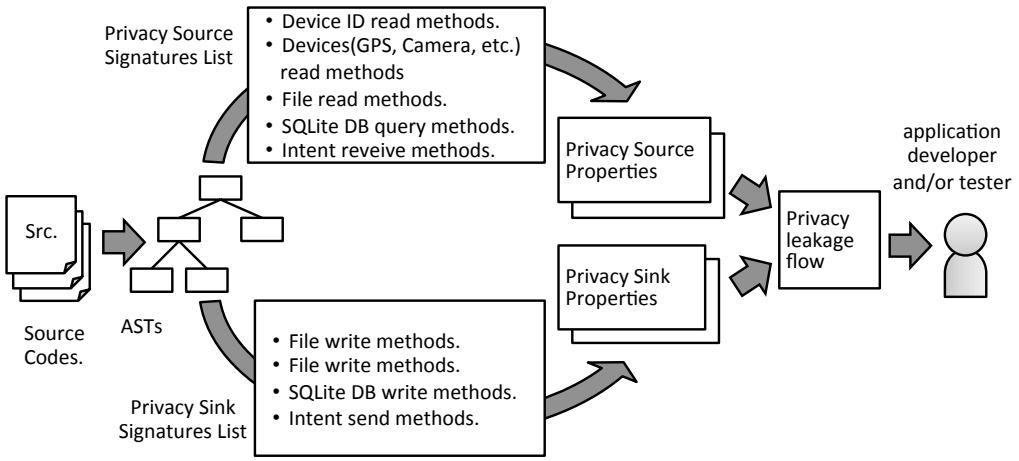

Figure 3: Schematics of security verification tool

target Android source files and build AST (Abstract Syntax Tree) with Eclipse JDT (Java Development Tools)[18] libraries.

Then identify the sources of critical information (terminal identification number, location information, etc) and sink of them (write to file stream, sending intent). Then verify the critical information flow.

4.2 Identifying source of critical information

Figure 4 shows the case brief example of identifying critical information source.

In this example, IMEI and Subscriber identification number are obtained.

1. Identifying method invocation.

   Scan nodes within the AST and pick a method invocation node (METHOD_INVOCATION) for further investigation.

2. Collate signature

   Collate its node to prepared signature list. Whether the node matched any prepared signature within, go next step, not match, return to scan nodes in AST.

3. Identify critical information

   Identify node containing the result of method invocation to identify the variable containing critical information. In this example, value of IMEI is substituted to variable `imei`, and value of IMSI is substituted to variable `imsi`.

4.3 Identify the sink of critical information

Figure 5 shows the case brief example of identifying critical information sink. In this example, IMEI and IMSI ob-

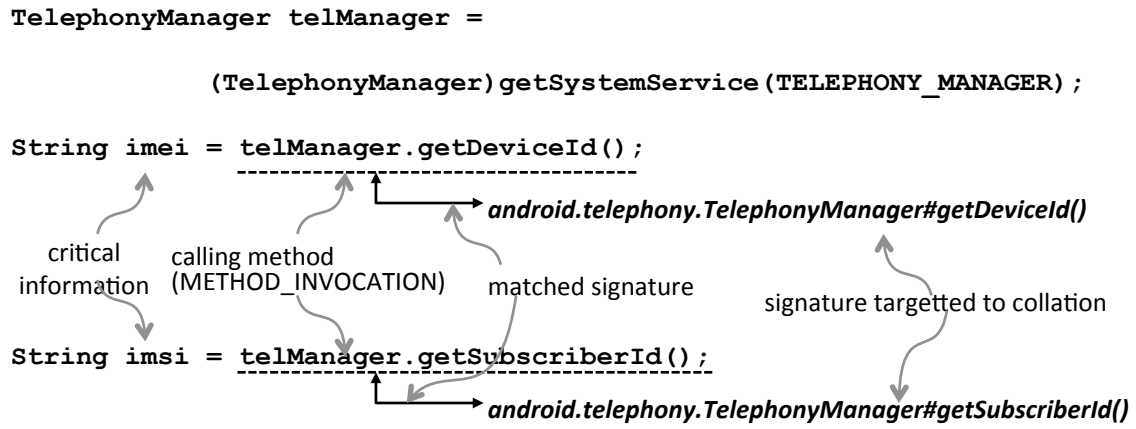

Figure 4: Private Information Source

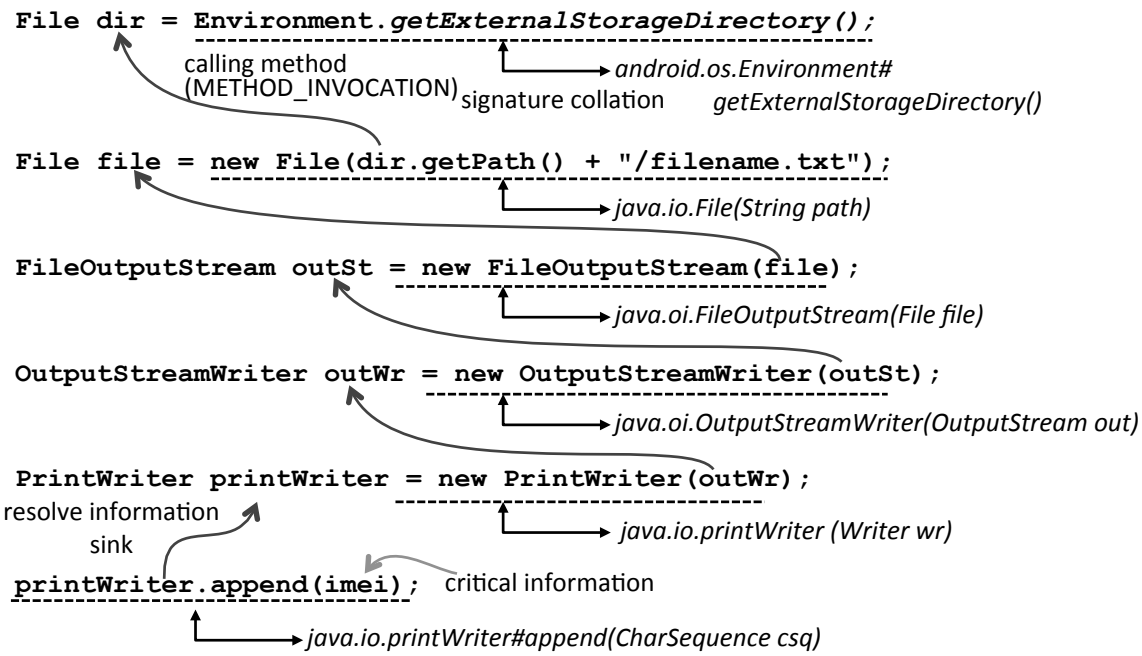

Figure 5: Private Information Sink

tained in the example shown in the last section, are saved in external memory card. Here, the procedure to identify the critical information sink is as below.

1. Identifying method invocation

   Scan nodes within AST and pick a method invocation node (METHOD_INVOCATION) and resolve the type of method invocation node with JDT API for further investigation.

2. Collate signature

   Collate the node with normalized signature list. Whether the node matched any prepared signature within, go next step, not match, return to scan nodes in AST.

3. Resolve parameters and leakage destination

   Resolve information sink and kind of critical information. In this example, resolve output destination of `PrintWriter` and obtain the directory path of output file. Thereafter check the parameters of append method belong to `PrintWriter`, resolve output information is critical or not.

5. API OBJECT OF VERIFICATION

There are many critical information handled in Android

Table 3: Methods retrieve Unique Identifier

Class	Method or Field	Identifier	Description
android.telephony. TelephonyManager	getDeviceID()	IMSI	Returns the unique subscriber ID, for example, the IMSI for a GSM phone.
	getSubscriberID()	IMEI	Returns the unique device ID, for example, the IMEI for GSM and the MEID or ESN for CDMA phones.
	getSimSerialNumber()	ICC	Returns the serial number of the SIM
	getVoiceMailAphaTag()	Voice mail ID	Retrieves the alphabetic identifier associated with the voice mail identifier.
	getVoiceMailNumber()	Voice mail number	Returns the voice mail number. Return null if it is unavailable.
	getLine1Number()	Telephone Number	Returns the phone number string for line 1, for example, the MSISDN for a GSM phone.
android.os.Build	SERIAL	Serial Number	A hardware serial number.
android.provider. Settings.Secure	ANDROID_ID	Android ID	A 64-bit number (as a hex string) that is randomly generated on the device's first boot and should remain constant for the lifetime of the device
android.net.wifi. WifiInfo	getMacAddress()	MAC address	MAC address assigned to WiFi device.
android.Bluetooth. BluetoothDevice	getAddress()	MAC address	Returns the hardware address of this Bluetooth Device.

environment. APIs treat them are listed in method signatures.

5.1 Input of critical information

1. Information for terminal/subscriber identification
   Information these are used as an identification numbers of terminal are below. These APIs must be checked as critical information source.
   - IMEI (International Mobile Subscriber Identity) reading API
   - IMSI (International Mobile Subscriber Identity) reading API
   - USIM serial number reading API
   - MAC address reading API
2. Devices
   There are some APIs for reading sensed device information as below.
   - Location information obtained from GPS
   - Compass information
   - Access to digital camera
3. Access to Storage
   There are many kind of storage access APIs, for variety of storage kind as below.
   - Reading from file system (including external storage card). item Access to SQLite database.
4. Inter-Application Communication
   There are some methods for inter-application communication as below.
   - Receiving implicit intent
   - Receiving explicit intent
   - Reading Content Provider

As a initial stage of prototyping, we focus on the unique and immutable (or almost immutable) user and/or terminal identifier values. These values are retrieved from the methods indicated in Table3[21].

5.2 Output Critical Information

1. Access to storage
   As shown below, these are APIs for writing to storage.
   - Writing to file system (including external storage card).
   - Access to SQLite database.
2. Inter-Application Communication
   There are some methods for inter-application communication. As shown in below.
   - Sending implicit intent
   - Sending explicit intent

6. PRESENTING VERIFICATION RESULT

Detected information flow that cause leakage of critical information, must be presented to developer as warning. Severity of warning is depend on the kind of critical information and output destination of it. However, as a feature of development tool, there are some restrictions.

- Unwritten code
  The case of Method called from target source.
- Undecidable information flow
  From the nature of static analysis, execution flow is undecidable.

In these case, present there is undecidable information flow, and warn there might be vulnerability.

7. CONCLUSION

In this paper, we proposed the tool for Android application developers/vendors to guarantee the security of Android application product.

We indicated that analyze an application about the harmlessness on privacy, privacy information move/transform flow analysis is required. Furthermore, the privacy information flow analysis is based on privacy sink/source properties analysis and flow analysis.

This tool might be utilized in situations as below.

- In an Android application design and/or test process, detecting vulnerability.
- In the commissioned software development style, receiving inspection.

Now we are designing and implementing this tool. After that, we evaluate it and obtain feedback to improve configurability and presenting format of analysis result.

8. REFERENCES

[1] Aho, A.V., Sethi, R. and Ullman, J.D., *Compilers: Principles, Techniques, and Tools*, Prentice Hall, New Jersey, 2006.

[2] Appel, A.W. and Palsberg, J., *Modern Compiler Implementation in Java*, Cambridge University Press, 2002.

[3] Asher, A., Skype-Skype Security blog- [Fixed] Privacy vulnerability in Skype for Android, 2011, http://blogs.skype.com/security/2011/04/privacy_vulnerability_in_skype.html

[4] Bugiel, S., Davi, L., Dmitrienko, A., Fischer, T., Sadeghi, A. and Shastry, B., Towards Taming Privilege-Escalation Attacks on Android, in *19th Annual Network & Distributed System Security Symposium*, (San Diego, CA, USA, 2012).

[5] Chaudhuri, A., Language-Based Security on Android, in *ACM SIGPLAN Fourth Workshop on Programming Languages and Analysis for Security*, (Dublin, Ireland, 2009). 1-7. DOI=http://dx.doi.org/10.1145/1554339.1554341

[6] Enck, W., Ongtang, M. and McDaniel, P., Understanding Android Security, *IEEE Security and Privacy*, Volume 7(1) . 50-54.

[7] Enck, W., Gilbert, P., Chun, B, Cox, L.P., Jung, J., McDaniel,P. and Sheth, A.N., TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones, in *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation*, (Vancouver, BC, Canada, 2010).

[8] Felt, A., Chin, E., Hanna, S., Song, D. and Wagner, D., Android permissions demystified, in *18th ACM Conference on Computer and Communications Security* (Chicago, IL, USA, 2011). 627-638. DOI=http://dx.doi.org/10.1145/2046707.2046779

[9] Gosling, J., Joy, B., Steele, G. and Bracha, G., *The Java Language Specification (Third Edition)*. Addison-Wesley, Boston, 2005.

[10] Hiroshi L., Android and Security, 2012, http://googlemobile.blogspot.jp/2012/02/android-and-security.html

[11] Hornyack, P., Han, S., Jung, J., Schechtery,S. and Wetherall, D., These aren't the droids you're looking for: retrofitting android to protect data from imperious applications, in *18th ACM Conference on Computer and Communications Security*, (Chicago, IL, USA, 2011). 639-652. DOI=http://dx.doi.org/10.1145/2046707.2046780

[12] Jing, Y., Ahn, G. and Hu, H., Model-Based Conformance Testing for Android, in *Advances in Information and Computer Security, 7th International Workshop on Security, IWSEC* (Fukuoka, JAPAN, 2012). Springer, 1-18.

[13] Marforio,C., Aurélien, F. and Capkun,S. 2011. *Application collusion attack on the permission-based security model and its implications for modern smartphone systems*, Technical Report. Eidgenössische Technische Hochschule Zürich. DOI=http://dx.doi.org/10.3929/ethz-a-006720730

[14] Oberheide, J., Dissecting, *Android's Bouncer*, 2012, https://blog.duosecurity.com/2012/06/dissecting-androids-bouncer/

[15] Schlegel,R. , Zhang,K. , Zhou, X., Intwala, M. , Kapadia, A. and Wang, X. 2011. *Soundcomber: A Stealthy and Context-Aware Sound Trojan for Smartphones*, in *18th Annual Network and Distributed System Security Symposium*, (San Diego, CA, USA, 2011).

[16] Android SDK — Android Developers, http://developer.android.com/sdk/index.html

[17] Android Security Overview, http://source.android.com/tech/security/index.html

[18] Eclipse Java development tools(JDT), 2012, http://www.eclipse.org/jdt/

[19] Google Play, https://play.google.com/store

[20] Manifest.permission — Android Developers, 2012, http://developer.android.com/reference/android/Manifest.permission.html

[21] Package Index — Android Developers, 2012, http://developer.android.com/reference/packages.html

[22] (permission) Android developers, 2012, http://developer.android.com/guide/topics/manifest/permission-element.html