Comnoid: Information Leakage Detection using Data Flow Analysis on Android Devices

Sunita Dhavale
Dept. of Computer Science and Engineering
Defence Institute of Advanced Technology,
Girinagar, Pune-411025

Bhushan Lokhande
Dept. of Computer Science and Engineering
Defence Institute of Advanced Technology,
Girinagar, Pune-411025

ABSTRACT
Security and privacy of Smartphone data are critical requirements in case of both personal as well as corporate environment. Hence, there is a need to come up with an effective solution in order to address data leakage issues in smartphones. Generally, taint analysis techniques are used for information flow tracking and data leakage detection purpose. Static Taint analysis techniques can detect the leakages that may not be exposed in runtime. Static analysis derives the information about program's behaviour by inspecting the program's code and discovering multiple paths of a program execution. In this work a static taint analysis tool Comnoid is proposed along with companion app ApkGrabber. Comnoid is based on open source tool FlowDroid and is capable of analyzing the inter app communication. Existing version of FlowDroid tool can provide precise static taint analysis but it lacks capability to analyze inter app communication between Android applications. Thus the aim of proposed scheme is to develop a tool to perform Static Taint analysis with inter app analysis which will take Android application APK files as an input and produce a data leakage report.

General Terms
Android Security, Operating System Security.

Keywords
Android Operating System; static and dynamic taint analysis; data flow analysis; Mobile Security

1. INTRODUCTION
Smartphone usage has exceeded those of desktops's in the recent years. At the same time, Security and privacy concerns about Smartphone data are increasing exponentially in case of both personal as well as corporate environment. Android being the most used Smartphone OS, needs solution to address information leakage issues. Today professionals prefer utilizing their personal Smartphone/Tablet for carrying out corporate work related tasks like email, docs, calendar, corporate apps etc. This helps them to achieve greater flexibility and balance between personal and corporate life. This trend belongs to well known Bring Your Own Device (BYOD) model. Companies have successfully adopted BYOD concept and are ready to manage security and privacy concerns as BYOD model promises more work efficiency of employees [1, 2, 23]. Android nears 80% market share in Global smartphone shipments [3, 4] and is also the biggest target of mobile malwares. These malwares pose high risk to the privacy of user's data. Many of the genuine applications have the permissions to access various local resources like contacts, mail, GPS, SD card, camera, microphone, accelerometer etc. They may leak data to unwanted parties on the internet, or other malicious applications on the phone. These security and privacy concerns are being addressed in current Android research domain. Taint analysis techniques are used for tracking the information flow and possible leakage on Android [7-14]. Taint analysis [6, 7, 23] is data flow analysis technique that can track flow of sensitive information. In this analysis, Taint sources and Taint sinks of sensitive data are predefined. In context of Android, Taint sources are Account, Email, Contact, Calendar, Database, File, Location Log, Phone State, SMS/MMS, Settings and Unique identifiers (IMEI) etc. Whereas Taint sinks are the points from where data can leak out of the system. Common taint sinks in Android are internet, publicly accessible storage i.e. memory card, inter process communication message and SMS transmission. Taint tracking discovers the route from source to sink if any exists. If source data reaches the sink, it is identified as instances of data leakage.

There are two approaches for taint analysis: Dynamic Taint analysis and Static Taint analysis. Based on dynamic taint analysis, many works have been proposed [8-10, 23]. Dynamic analysis techniques monitor the code as it is being executed and based upon runtime information. It suffers from performance overheads in case of real time monitoring of applications. Although, it provides detailed information of specific run, it cannot provide complete overview of all possible execution path of program. Further, for real time processing, this approach needs to be implemented on actual Android device or Virtual Android device.

In contrast to dynamic taint analysis, static taint analysis techniques considers all possible execution paths of the program [11-15]. It analyzes complete code without the need of execution of the code; and creates control flow graph (CFG) of program. CFG is used to trace the flow of sensitive information from sources to sinks. Modern static taint analysers convert programs code into some intermediate representation, which can be effectively processed to generate CFG and Call graphs [13, 23]. Static analysis takes more time to analyse the program than dynamic analysis as it processes complete code and all execution paths. But it has no real time performance overhead as processing is done statically before the code is actually being executed. Static taint analysis on Android can be performed by extracting android package (apk files) of all installed applications and then processing them. Also it can be done at application market level like Google Play Store.

In this paper, a static taint analysis tool Comnoid is proposed along with companion app ApkGrabber. Comnoid is based on open source tool FlowDroid and is capable of analyzing the inter app communication. A companion app ApkGrabber can be used for collecting application APKs from smartphones. The remaining part of the paper is organized as; Section II provides the overview of flowdroid. Proposed scheme with experimental results are explained in Section III followed by Conclusions in Section IV.

2. OVERVIEW OF FLOWDROID

Christian Fritz et al. [13] presented FlowDroid, precise static taint analysis for android applications. It takes the Android apk file as input for processing and does the static taint analysis. This approach models complete Android lifecycle [6] precisely to handle callbacks. It is context sensitive as well as flow, field and object sensitive. Source and sink for targeted Android version are identified by using SuSi framework [17]. Detection of source, sink and entry-point is done by parsing manifest, dalvik executable (dex) and xml layout files extracted from application apk.

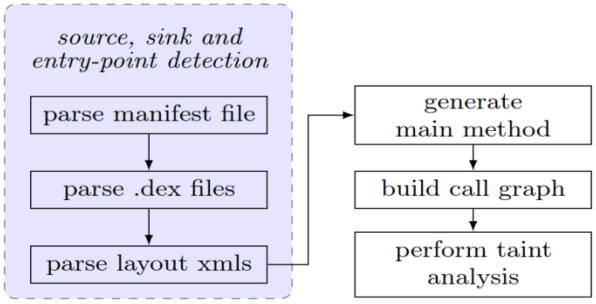

Fig 1: Overview of FlowDroid [10].

FlowDroid generates dummy main method from parsed data as shown in Fig 1. Taint analysis is performed on this graph to discover paths from source to sink; all the discovered paths are then reported. IFDS framework is used to formulate inter procedural data flow analysis problem which creates exploded super graph [18]. FlowDroid ignores dynamic loading and reflection. It treats JNI code as black box and explicit taint propagation rules are defined for common native methods. It lacks taint tracking for intra-app and inter-app communication. Also the current implementation of FlowDroid ignores reflexive calls and dynamically loaded codes.

Along with FlowDroid, the authors have also proposed DroidBench which is very Android specific test suite containing set of vulnerable apps. On DroidBench evaluation FlowDroid outperformed other commercially available tools. It detected all seven data leaks of Insecure Bank App (vulnerable app used for evaluation purpose) [19]. It also performed well on Java specific benchmark suites. FlowDroid is highly precise static taint analysis tool, recently it is been improved to support implicit flows [22] and it is available as an open source. It can be executed on any computer with Android application apk's are given as input for further analysis [13]. Further in [25], the security challenges of Android communication from the perspectives of Intent senders and Intent recipients are given in more details.

3. PROPOSED SCHEME

The proposed static taint analysis tool 'Comnoid' consists of two major modules; Comnoid and ApkGrabber as shown in Figure 2. The details of both modules are explained in following sections.

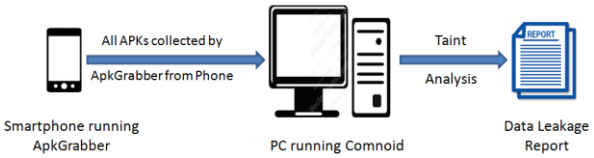

Fig 2: Processing scenario of Comnoid along with ApkGrabber

3.1 Comnoid

Comnoid is based on FlowDroid open source tool. It takes Android application apk file as input and extracts Dex file, Manifest file and Xml layout files for processing as shown in Figure 3.

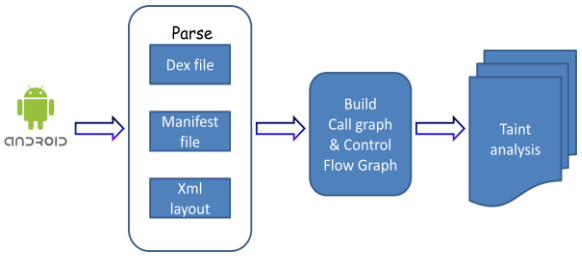

Fig 3: Comnoid overall processing architecture

Manifest file is used to extract metadata about various Android components as well as permissions used in it. Dex file contains dalvik executable code which will be disassembled for call graph and Control flow graph generation (CFG). Xml layout files contains the application UI components i.e. Android Activities. Soot framework is used to generate CFG. It has dexpler tool which disassembles dalvik code and generates Jimple format code. Jimple code is used to form Jimple CFG & call graph. This control flow graph is processed to trace propagation of taint source (sensitive information) to the taint sink (data leak gateway). If the link found from taint source to taint sink in the CFG, it is reported as data leak instance in the final report.

A new inter App processing module is added in existing Flowdroid framework which is responsible for inter application communication processing and finding the data leakages as shown in Fig. 4. In Android inter app communication is performed via Binder responsible for Inter Process Communication (IPC). Communication messages between applications as well as components are in the form of Intent [24]. Intent analysis is performed to find out data leakages in Inter app communication. This module contains three sub modules named Permission mapper, Intent Analyser, Component Analyser as shown in Fig. 4.

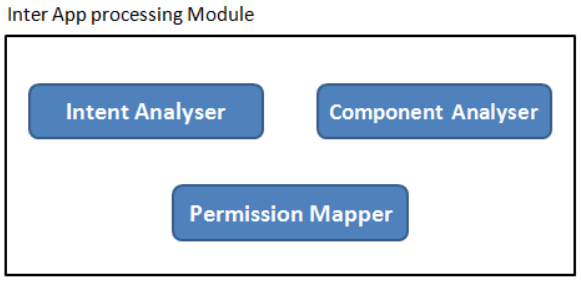

Fig 4: Inter App processing sub modules

3.1.1 Permission Mapper

Permission Mapper extracts the permissions used by application. These extracted permissions can belong to following categories as shown in Figure 5.

Normal Permissions- These are basic permission which are granted automatically by Android system.

Dangerous Permissions- User grants these permissions to application at the time of installation. If denied, the application will not be installed.

Signature Permission- granted only to the application which is developed by same developer who defined the permission. Useful when set of applications developed by same developer wants to communicate.

Signature System Permission- Application pre-installed by System manufacturer can have these permissions.

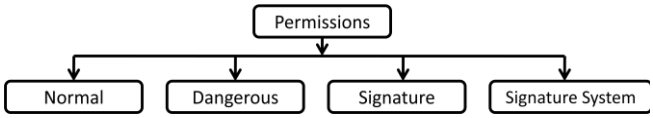

Fig 5: Permission Mapper

Normal and Dangerous permissions are used for further processing as signature and signature system permissions does not lead to the data leakage. Normal permissions are extracted from Android manifest and dangerous permissions are extracted from Application manifest file.

3.1.2 Intent Analyzer Module

Intent analyser tracks the intent when it is passed as parameter or returned as value of any function. Figure 6 shows the details of Intent Analyzer Module. Explicit intents are sent to the exact intended application but implicit intent may be received by malicious application. Such intents are tracked from source to sink in the control flow graph. Thus data leak instance is generated when implicit intent with weak or no permission requirements is traced. For each such intent, its action, flags and data are tracked.

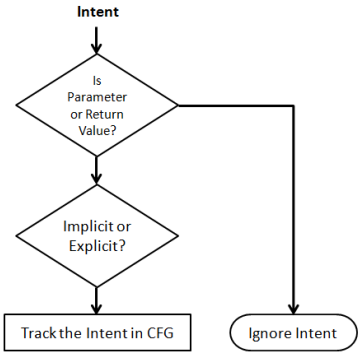

Fig 6: Intent Analyser

```
// Create the text message with a string
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
sendIntent.setType(HTTP.PLAIN_TEXT_TYPE); // "text
                    /plain" MIME type


// Verify that the intent will resolve to an activity
if (sendIntent.resolveActivity(getPackageManager()) !
                    = null) {
  startActivity(sendIntent);
}
```

The code shows example of implicit Intent, in this intent the plain text data embedded. Such intent can be received by different apps like sms, email and whatsapp like chatting apps. In such case, sendIntent which contains sensitive information textMessage string is tracked.

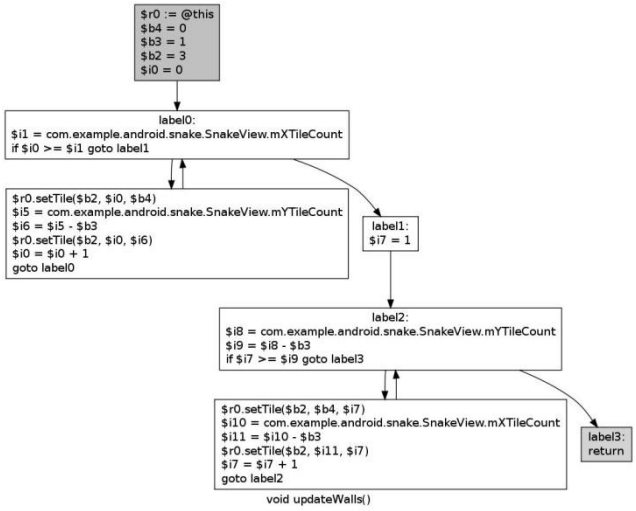

Fig 7: Example of Jimple Control Flow graph

After identifying the intent for tracking, it is tracked in the generated Jimple CFG. Figure 7 shows the Jimple CFG for SnakeView app. '$' symbol represents it's a variable, and suppose

$r0 = sendIntent

$r0 is tracked through the CFG and check whether it passes to the taint sink.

3.1.3 Component Analyzer Module

Component Analyser extracts components information from manifest file as well as from translating Dalvik instructions. This information is checked to see if EXPORTED flag is set or intent filters are present. If either of this is found the component is identified as exported component and further traced for data leakage. Exported component with no permission or weak permission check are reported as data leak. If Comnoid finds a exported component that registers to receive Intents with a non-Android action string and also finds components that transmit implicit Intents with the same action string, Comnoid issues a data leak. Exported broadcast receivers reported for broadcast injection leak, exported activities for activity launch attack and exported services for service launch attack.

3.2 Companion Android App ApkGrabber

A companion Android app is developed which creates apk files for all the installed applications on the smartphone and store it in one destination folder on SD card. ApkGrabber requests the Android package manager details and list of all applications on phone. From these details it tracks the application folder and generates the apk file. Phone can be connected to the Pc and Comnoid will given the address of that folder which contains all the apk's. Comnoid will the generate the data leakage report for all the applications. This fastens the manual process of getting APK file from the marketplace and sometimes user may have installed applications other than market store of android i.e. Google Play Store.

Companion app stores all apk in the sdcard/APK folder. This folder's path is given to the Comnoid to analyse all the apk's and generate report.

4. EXPERIMENTAL RESULTS

The data leakage reports generated from Flowdroid and Comnoid are compared. This output data leak report can be analyzed by technicians and/or programmers to detect number of vulnerabilities. Following are some of the sample data leakage reports generated for the ArraysAndList, Callbacks and InterApp related vulnerability apps from DroidBench benchmark using Flowdroid.

1. droidbench\ArraysAndLists_ArrayAccess1.apk

RESULT SIZE: 1

#Found a flow to sink virtualinvoke $r2.<android.telephony.SmsManager: void

SendTextMessage(java.lang.String,java.lang.String,java.lang.String,android.app.PendingIntent,android.app.Pe

ndingIntent)>("+49 1234", null,$r7, null, null), from the following sources: - virtualinvoke

$r6.<android.telephony.TelephonyManager: java.lang.String getDeviceId()>() (in <de.ecspride.ArrayAccess1:

void onCreate(android.os.Bundle)>)

2. droidbench\Callbacks_Button2.apk

RESULT SIZE:4

#Found a flow to sink staticinvoke <android.util.Log: int i(java.lang.String,java.lang.String)>("TAG", $r3),

from the following sources: - virtualinvoke $r5.<android.telephony.TelephonyManager: java.lang.String

getDeviceId()>() (in <de.ecspride.Button2: void clickOnButton3(android.view.View)>)

#Found a flow to sink staticinvoke <android.util.Log: int i(java.lang.String,java.lang.String)>("TAG", $r3),

from the following sources: - virtualinvoke $r5.<android.telephony.TelephonyManager: java.lang.String

getDeviceId()>() (in <de.ecspride.Button2: void clickOnButton3(android.view.View)>)

#Found a flow to sink staticinvoke <android.util.Log: int i(java.lang.String,java.lang.String)>("TAG", $r3),

from the following sources: - virtualinvoke $r5.<android.telephony.TelephonyManager: java.lang.String

getDeviceId()>() (in <de.ecspride.Button2: void clickOnButton3(android.view.View)>)

#Found a flow to sink virtualinvoke $r2.<android.telephony.SmsManager: void

sendTextMessage(java.lang.String,java.lang.String,java.lang.String,android.app.PendingIntent,android.app.Pe

ndingIntent)>("+49 1234", null, $r5, null, null), from the following sources: - virtualinvoke

$r5.<android.telephony.TelephonyManager: java.lang.String getDeviceId()>() (in <de.ecspride.Button2: void

clickOnButton3(android.view.View)>)

3.droidbench\InterAppCommunication_ActivityCommunication1.apk

No results found.

4. droidbench\InterAppCommunication_IntentSink1.apk

No results found.

5. droidbench\InterAppCommunication_IntentSink2.apk

No results found.

From results, it is found that the .apk files containing inter app vulnerabilities are not recognized by Flowdroid. Following are some of the sample data leakage reports generated for the ArraysAndList, Callbacks and InterApp related vulnerability apps from DroidBench benchmark using Comnoid.

1. droidbench\ArraysAndLists_ArrayAccess1.apk

RESULT SIZE: 1

#Found a flow to sink virtualinvoke $r2.<android.telephony.SmsManager: void

SendTextMessage(java.lang.String,java.lang.String,java.lang.String,android.app.PendingIntent,android.app.Pe

ndingIntent)>("+49 1234", null,$r7, null, null), from the following sources: - virtualinvoke

$r6.<android.telephony.TelephonyManager: java.lang.String getDeviceId()>() (in <de.ecspride.ArrayAccess1:

void onCreate(android.os.Bundle)>)

2. droidbench\Callbacks_Button2.apk

RESULT SIZE:4

#Found a flow to sink staticinvoke <android.util.Log: int i(java.lang.String,java.lang.String)>("TAG", $r3),

from the following sources: - virtualinvoke $r5.<android.telephony.TelephonyManager: java.lang.String

getDeviceId()>() (in <de.ecspride.Button2: void clickOnButton3(android.view.View)>)

#Found a flow to sink staticinvoke <android.util.Log: int i(java.lang.String,java.lang.String)>("TAG", $r3),

from the following sources: - virtualinvoke $r5.<android.telephony.TelephonyManager: java.lang.String

getDeviceId()>() (in <de.ecspride.Button2: void clickOnButton3(android.view.View)>)

#Found a flow to sink staticinvoke <android.util.Log: int i(java.lang.String,java.lang.String)>("TAG", $r3),

from the following sources: - virtualinvoke $r5.<android.telephony.TelephonyManager: java.lang.String

getDeviceId()>() (in <de.ecspride.Button2: void clickOnButton3(android.view.View)>)

#Found a flow to sink virtualinvoke $r2.<android.telephony.SmsManager: void

sendTextMessage(java.lang.String,java.lang.String,java.lang.String,android.app.PendingIntent,android.app.Pe

ndingIntent)>("+49 1234", null, $r5, null, null), from the following sources: - virtualinvoke

$r5.<android.telephony.TelephonyManager: java.lang.String getDeviceId()>() (in <de.ecspride.Button2: void

clickOnButton3(android.view.View)>)

3.droidbench\InterAppCommunication_ActivityCommunication1.apk

RESULT SIZE:1

#Found a flow to sink virtualinvoke $r2.<android.telephony.SmsManager: void

sendTextMessage(java.lang.String,java.lang.String,java.lang.String,android.app.PendingIntent,android.app.Pe

ndingIntent)>("+49", null, $r3, null, null), from the following sources: - virtualinvoke

$r4.<android.telephony.TelephonyManager: java.lang.String getDeviceId()>() (in <de.ecspride.Activity2: void

onCreate(android.os.Bundle)>)

4. droidbench\InterAppCommunication_IntentSink1.apk

#Found a flow to sink virtualinvoke $r0.<de.ecspride.IntentSink2: void

startActivity(android.content.Intent)>($r3) on line 22, from the following sources: - virtualinvoke

$r6.<android.telephony.TelephonyManager: java.lang.String getDeviceId()>() (in <de.ecspride.IntentSink2:

void startIntent(android.view.View)>)

5. droidbench\InterAppCommunication_IntentSink2.apk

RESULT SIZE:1

#Found a flow to sink virtualinvoke $r0.<de.ecspride.IntentSink2: void

startActivity(android.content.Intent)>($r3) on line 28, from the following sources: - virtualinvoke

$r6.<android.telephony.TelephonyManager: java.lang.String getDeviceId()>() (in <de.ecspride.IntentSink2:

void startIntent(android.view.View)>)

From results, it is found that, Comnoid is able to detect the interapp vulnerabilities successfully. Table 1 gives the comparison between Flowdroid and Comnoid.

Current version of Comnoid does not support the detection of privilege delegation through Pending Intents and Intents that carry URI read-write permissions. Pending Intent is a token that one application (say App1) gives to another application (say App2). This allows App2 to inherit permissions of App1 to execute a predefined piece of code. In future, Comnoid tool will be enhanced to trace back to the original intents.

Table 1: Comparison between Flowdroid and Comnoid

Apk	Leaks	FlowDroid Result	Comnoid Results
Callbacks_ MethodOverride 1.apk	1	Success	Success
Callbacks_ MultiHandlers1. apk	1	Fail	Success
Callbacks_ Ordering1.apk	2	Success	Success
Callbacks_ RegisterGlobal1 .apk	1	Success	Success
Callbacks_ Unregister1.apk	1	Success	Success
InterAppComm unication_Intent Sink1.apk	1	Fail	Success
InterAppComm unication_Intent Sink2.apk	1	Fail	Success
InterAppComm unication_Activ ityCommunicati on1.apk	1	Fail	Success

5. CONCLUSIONS

Smart phones can contain private and confidential data, when allowed for accessing corporate applications. Malicious apps can steal such sensitive data or can track users without their consent. Considering pros and cons of both approaches, it is found that static taint analysis techniques are best suited for analysis on Android Market site and dynamic taint analysis for real time tracking on Android phone. Thus static taint-analysis tools for Android applications can be used to detect illegal data leakage before any malicious application executed or even before its installation.

Many current static taint analysis approaches do not analyse dynamically loaded code and Inter-app communication. There are many data leakage issues in inter-app communication. From results, it is found that proposed Comnoid tool can detect inter-app data leakage instances for DroidBench benchmark successfully. Comnoid gives similar results as FlowDroid with additional support for inter app analysis. An ApkGrabber companion Android app is also developed to collect apk's of installed application on phone. Thus Comnoid can be efficiently used for data leak detection for Android application on Market store as well as for individual user's smartphone applications.

Comnoid suffers with the drawbacks which are common to that of other static analysis tools like it is not able to analyse the reflexive calls and dynamically loaded code.

6. REFERENCES

[1] Keith W. Miller, Jeffrey Voas, George F. Hurlburt, "BYOD: Security and Privacy Considerations", IT Professional (Volume:14,Issue:5), Sept.-Oct. 2012, pp.53-55.

[2] Antonio Scarfò, "New security perspectives around BYOD", Seventh International Conference on Broadband", Wireless Computing, Communication and Applications 2012

[3] International Data Corporation Press Release, http://www.idc.com/getdoc.jsp?containerId=prUS24257413, Aug. 07, 2013 [Oct. 24, 2013]

[4] Technology Research Gartner Inc, http://www.gartner.com/newsroom/id/2573415, Aug. 14, 2013 [Oct. 24, 2013]

[5] Android Security Overview, https://source.android.com/devices/tech/security/index.html [Oct. 24, 2013]

[6] Android Activity Lifecycle, http://developer.android.com/guide/components/activities.html [Oct. 24, 2013]

[7] Edward J. Schwartz, Thanassis Avgerinos, David Brumley, "All You Ever Wanted to Know About Dynamic Taint Analysis and Forward Symbolic Execution (but might have been afraid to ask)" SP '10 Proceedings of the 2010 IEEE Symposium on Security and Privacy, pp.317-331

[8] William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, Anmol N. Sheth. "TaintDroid: An Information-Flow Tracking System for Real-time Privacy Monitoring on Smart phones" 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI' 10) 2010.

[9] Peter Hornyack, Seungyeop Han, Jaeyeon Jung, Stuart Schechter, and David Wetherall, "These Aren't the Droids You're Looking For", Retroffiting Android to Protect Data from Imperious Applications In Proc. of ACM CCS, October 2011

[10] Daniel Schreckling, Johannes Kostler, Matthias Schaff, "Kynoid: Real-time enforcement of fine-grained, user-defined, and data-centric security policies for Android", information security technical report 17, pp.71-80, 2013

[11] Zhemin Yang and Min Yang, "LeakMiner: Detect Information Leakage on Android with Static Taint Analysis", In Software Engineering (WCSE), 2012 Third World Congress on, pp.101–104, 2012

[12] Zhibo Zhao and F.C.C. Osono, "Trustdroid: Preventing the use of smartphones for information leaking in corporate networks through the used of static analysis taint tracking". In Malicious and Unwanted Software (MALWARE), 7th International Conference on, pages 135–143, 2012

[13] Christian Fritz, Steven Arzt, Siegfried Rasthofer, Eric Bodden, Jacques Klein, Alexandre Bartel, Yves le Traon, Damien Octeau, Patrick McDaniel, "Highly Precise Taint Analysis for Android Applications", EC SPRIDE Technical Report. Nr. TUD-CS-2013-0113. May, 2013

[14] Clint Gibler, Jonathan Crussell, Jeremy Erickson, and Hao Chen, "AndroidLeaks: automatically detecting potential privacy leaks in android applications on a large scale", Proceeding TRUST'12 Proceedings of the 5th international conference on Trust and Trustworthy Computing pp.291-307, 2012

[15] Jinyung Kim, Yongho Yoon, Kwangkeun Yi, Junbum Shin, "ScanDal: Static Analyzer for Detecting Privacy Leaks in Android Applications" MoST 2012: Mobile Security Technologies, May 2012

[16] Golam Sarwar (Babil), Olivier Mehani, Roksana Boreli, Mohamed-Ali Kaafar, "On the Effectiveness of Dynamic Taint Analysis for Protecting Against Private Information Leaks on Android-based Devices", SECRYPT, 10th International Conference on Security and Cryptography 2013

[17] Steven Arzt, Siegfried Rasthofer, and Eric Bodden. "SuSi: A Tool for the Fully Automated Classification and Categorization of Android Sources and Sinks". Technical report, EC SPRIDE Technical Report TUD-CS-2013-0114, 2013

[18] Thomas Reps, Susan Horwitz, and Mooly Sagiv. "Precise interprocedural dataflow analysis via graph Reachability", In POPL '95, pp.49–61, 1995

[19] Paladion. Insecurebank test app. http://www.paladion.net/downloadapp.html [Oct. 25, 2013]

[20] ANTLR, http://www.antlr.org/ [Oct. 25, 2013]

[21] Jasmin, http: //jasmin.sourceforge.net/guide.html [Oct.25, 2013]

[22] FlowDroid Now Supports Implicit Flows, http://sseblog.ec-spride.de/2013/10/flowdroid-implicit-flows/ Oct. 01, 2013 [Oct. 25, 2013]

[23] B Lokhande, S Dhavale, "Overview of information flow tracking techniques based on taint analysis for android", International Conference on Computing for Sustainable Global Development (INDIACom), 5-7 March 2014, New Delhi, Publisher: IEEE, pp. 749 – 753, 2014

[24] http://developer.android.com/reference/android/app/PendingIntent.html [Oct. 25, 2013]

[25] Erika Chin, Adrienne Porter Felt, Kate Greenwood and David Wagner, "Analyzing Inter-Application Communication in Android", In the Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services (MobiSys 2011), Bethesda, MD, USA, June 28 - July 01, 2011