Android Permission Re-Delegation Detection and Test Case Generation

Jiagui Zhong, Jianjun Huang, Bin Liang
School of Information, Renmin University of China, Beijing 100872, China
{zhongjg, hjj, liangb}@ruc.edu.cn

Abstract—**As smart phones are becoming widespread over the world, relevant security problems emerge. On Android platform, some applications are granted to access some restrictive resources via system APIs. Such applications may expose this capability to the other applications without certain permissions. This will lead to permission re-delegation attacks. In this paper, we describe how this vulnerability occurs on Android through inter-process communication (IPC). We focus on a major IPC channel in Android operating system, the intent based IPC. In order to help developers decrease the possibility of their applications to be attacked, we present a static analysis tool Diordna in this paper. Diordna works on Java bytecodes and finds out possible permission re-delegations from public entry points of applications. Diordna also leverages a dataflow analysis to generate intent oriented test case specifications, namely, to infer what should be contained in an intent object by which the target application will re-delegate its granted permissions. We have experimented our solution and Diordna on two pre-installed Android applications and it generates reasonable test case specifications that can be used to write testing programs.**

Keywords-Android permission re-delegation, detection, intent, IPC, test case generation

I. Introduction

The smart phone market has proliferated dramatically in recent years. According to some recent Nielsen reports, about 40% of US mobile phone users owns smart phones and 40% of these phones are with Android Operation System, which claims the largest share of US smart phone market [1][2]. The proliferation of Android OS market has also promoted researchers to work on it as it provides a different permission policy with other PC operation systems and different to iOS app market, there are numerous alternative app markets for users to obtain third-party applications for their Android phones.

A lot of researchers focus on applications that may lead to privacy and data leakage [3][4], such as phone IMEI number, location information, etc. They also propose solutions to provide provenance for smartphones [5][6]. Such applications may not be malicious and they could be system applications provided with the Android operating system itself. Cannon [7] and Jiang [8] found a data stealing vulnerability in the pre-installed Browser application in Android 2.2 and 2.3 separately. Researchers also pay attentions on malicious applications detection on Android. Jiang and his team have found a large number of Trojans in different Android apps markets [8][9]. Some studies also have been done to detect vulnerabilities of Android applications. Enck et al [10] provided a solution to detect vulnerabilities in the applications using a traditional commercial tool with their own analysis specification and they found some intent injection attacks in the studies applications, something relating to our intent oriented permission re-delegation. Chin et al [11] presented ComDroid to detect Intent related vulnerabilities, such as broadcast theft, activity hijacking, etc. ComDroid does not aim to detect permission re-delegation but it gives recommendations that may help developers to reduce such flaws. On the basis of ComDroid, Felt et al [12] presented a static analysis tool Stowaway to detect overprivilege in applications. However, requiring more permissions than needed would not lead to exploitable vulnerabilities as they only exist in the manifest file and have nothing to do with the source code. Felt et al also presented permission re-delegation attacks and defenses on Android [13]. They mainly focused on how to protect the system from relevant attacks and we focus in this paper mainly on the test case specification generation. Grace et al presented Woodpecker [14] to detect so-called capability leaks, aliased with permission re-delegation. Woodpecker works on the Dalvik bytecode directly and checks both explicit and implicit capacity leaks.

In this paper, we focus on detecting permission re-delegation and test case specification generation for Android applications. On Android platform, Intent is used for communications between either components in an application or applications. It constitutes a major part of Android IPC. Usually, an intent object holds an abstract description of an operation to be performed or a description of something that has happened or is being announced [15]. Due to the security mechanism of Android, an application must apply relevant permissions when it is installed. The installation fails if at least one of its required permission is denied by the user. However, through the inter-process communication (IPC) mechanism, one without specific permissions can trigger those method calls protected by such permissions. Then, permission re-delegation attack occurs (Fig. 1).

Our goal is to provide specifications for IPC unit testing. We present Diordna in this paper which conducts the detection and test case specification generation.

The rest of this paper is organized as follows. Section II discusses our solution for detection and test case generation in details. Section III presents two examples we use to evaluate our prototype and section IV concludes this paper.

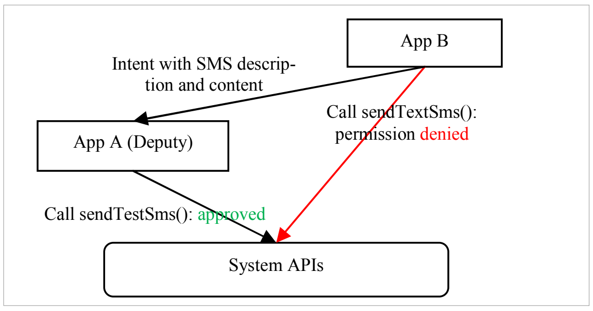
App B
Intent with SMS description and content
Call sendTextSms(): permission denied
App A (Deputy)
Call sendTestSms(): approved
System APIs

Figure 1. Android permission re-delegation example. App B can send a short message indirectly through App A's interface. sendTextMessage() is a critical system API that is protected by permission "android.permission.SEND_SMS".

II. System Design

We provide Diordna to detect permission re-delegation vulnerabilities in applications and generate IPC test cases for critical system APIs. Android applications are compiled into the Dalvik Executable (DEX) format and run in Dalvik virtual machine. We leverage a tool dex2jar [16] to transform DEX into standard JVM bytecode if we do not have the source code. The information we need in the analysis are maintained during the transformation, then we can utilize some existed JVM based tools to conduct the analysis. We choose WALA [17] as base of our tool. WALA works on either JVM bytecode or Java source code. Considering our detection purpose, bytecode is enough for analysis and much simpler than working on source code. In addition, both transformations from DEX to JVM bytecode and from JVM bytecode to source code may fail according to previous work [3][10]. Therefore, the transformation to source code is unnecessary. Leveraging WALA, Diordna constructs a callgraph of a given application and a control flow graph for each method to perform dataflow analysis that eventually generates test cases. Fig. 2 shows that overall process.

A. Permission Re-Delegation Detection

Due to the nature of Android applications and their runtime environment restrictions, one can only interact with another application through IPC, mainly by Intents. Hence, entry points are limited to those methods that can be invoked by the system in an inter-process Communication. Take BroadcastReceiver component for example, an implemented receiver is instantiated if the system passes an Intent object to it. Method onReceive() is invoked after the instantiation and it accepts an intent as input argument. This method is an entry point, as well as onStartCommand() and onCreate() in a Service component.

With given entrypoints information, Diordna builds a list of WALA entrypoint objects and passes it to WALA. WALA constructs the class hierarchy according to these entrypoints instead of its default main() entry point for traditional Java applications. Then WALA builds a callgraph and Diordna traverses on it to find out all the methods. When a system API is encountered, it checks whether it is contained in the permission map [12]. Protection level for each API is ignored in our design. Namely, the detector reports a permission re-delegation as long as it matches a system API in the permission map. In this process, Diordna constructs a graph to record all the methods that have been traversed and their hierarchy. When all re-delegation points are found out, Diordna traverses the graph reversely from such points to entrypoints. It aims to mark which methods is on the path from an entrypoint to a critical system API. Then, it is able to output all the exploitable paths.

B. Test Case Generation

With the paths from entry points to restrictive system APIs, Diordna can then perform a dataflow analysis to generate Intent oriented test cases. Intent oriented test case generation means that out work here just starts from those entry points that accepted an Intent object as input argument. We aim to infer the abstract description of the Intent. Such entrypoints include onStartCommand() in a service component and onReceive() in a broadcast component. For those entry points that do not accept an Intent object as input (e.g. onCreate() in a service component) or that do not interact with other applications through intents (e.g. interfaces in a content provider) are ignored temporarily in Diordna.

To a certain extent, the Intent oriented test cast generation is accompanied with taint propagation analysis. WALA provides control flow graph with Shrike [18] intermediate representations (IR) as alternative instructions. Shrike IR is a modified SSA form of standard JVM bytecode. Starting from a certain entrypoint, the generator marks its Intent parameter as a tainted source. Then the generator performs a flow sensitive and inter-procedural dataflow analysis.

Taking into consideration that Intent class itself provides a lot of interfaces to operate its private fields, we have to do semantic resolution for get*XXX*() methods in order to infer its inner data. Important fields include *action*, *data*, *extra*, etc. Besides, there are some functions in fundamental Java library we have to resolve. A typical example is the functions of class String. Our investigation to the system applications shows that equals() method is commonly used to determine if the inner data in the input intent object fits some conditions. We model

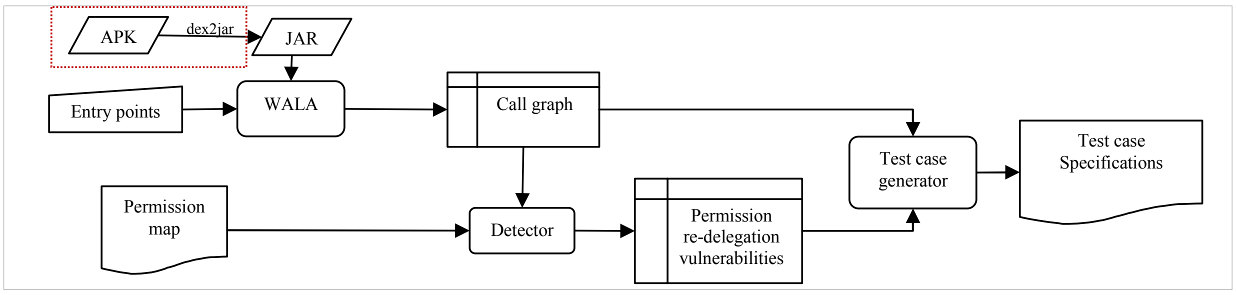

Figure 2. Overview of the system. With source code of an application, the dex2jar step is unnecessary

this method in the tool.

After we model those functions we are interested in, Diordna performs a path sensitive dataflow analysis. The analysis is inter-procedural because an entry point function normally does few things. Most actions are done in callee functions. We mainly pay attention to how a tainted source, namely the input Intent parameter of an entry point function, reaches a critical system API. We use traditional taint analysis method to trace the propagation of tainted values. Two types of instructions are handled: *conditional* and *invoke*. The former one is used to generate constraints that inputted Intent should be satisfied in order to trigger a specific protected system API. The latter one may refer to a critical API, a modeled function or an application function.

C. Limitations

According to previous studies, transformation from DEX format to JVM bytecode may fail at a relatively high possibility, about 20% [3]. When we began to test Diordna in a large scale, we will encounter this problem as we do not have source code of those applications that we get from various app markets.

Though WALA has implemented a lot of features that ease static analysis on bytecode level, it still cannot generate a fully completed callgraph for an Android application due to those implicit call flows. For example, listeners would not be connected to a certain call site in static analysis though they would be invoked at some places at runtime. In addition, subclasses of Thread implement the run() method and this method should follow a start() call. Such implicit flows cannot be handled as expected in WALA without any prior knowledge and it makes the analysis incomplete. Our strategy on handling conditional instructions seems to work well on the system applications but it may fall into some conditions in which we can't generate a full and right description of the Intent object.

III. Experimental Result

In this section, we discuss our evaluation of Diordna on two system application. We select Music and DeskClock as experiment objects, MediaPlaybackService and AlarmKlaxon separately in particular. Both have been reported to be vulnerable to suffer permission re-delegation attacks [19][20].

Music: The MediaPlaybackService itself was vulnerable in some older versions because as a publicly accessed service component, it exposed some protected APIs through an accessible interface, onStartCommand(). In such case, malicious applications could control the music player by simply sending specific intent objects through IPC channel. On Android 2.3.3, this component is protected for private use, but it registers some inner broadcast receivers which are not configured in the manifest file. Attackers can still control the music player by leveraging these receivers.

According to the permission map [12], android.media.MediaPlayer.start should be protected by permission android.permission.WAKE_LOCK if setWakeMode() has been called first, just as what is done in the Music. Our tool finds out several paths from the unproteced onReceive() entry-point to this restrictive system API. We present a test case in Fig. 3.

```
Intent {
   Extra: [ command -> {
      [ExtraType = StringExtra]
      @ == previous;
  }];
}
                                        (a)
Intent intent = new Intent ();
intent.setAction("com.android.music.musicservicecommand.previous");
intent.putStringExtra ("command", "previous");
sendBroascast (intent);
                                        (b)
```

Figure 3. Test case specification for Music application (a) and a real test case guided by the specifications (b). It is for the system API: android.media.Media-Player.start. Symbol '@' means the current object.

DeskClock: The permission re-delegation vulnerability in this application may lead to endless alarm as described in [20]. We present a callgraph in Fig. 4. Take the vibrate() method call for example, Diordna outputs a specification as in Fig. 5 (a). We can construct a test case guided by this specification. The Intent object should have a Parcelable *extra* whose key is "intent.extra.alarm" and whose value is an instance of class Alarm in the application. The instance value should be non-null and its fields satisfy the constraints separately. Alarm.alert could be null; Alarm.silent can be zero and Alarm.vibrate should not be zero. According to the source code, fields "silent" and "vibrate" hold Boolean values, so "zero" is converted to "false". In order to invoke the application and its interface, we need an action value from the application's manifest file; then we can build a test case shown Fig. 5 (b).

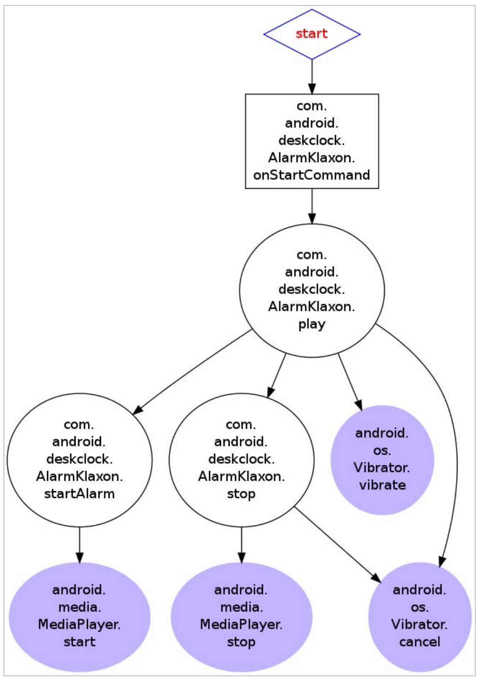
start
com. android. deskclock. AlarmKlaxon. onStartCommand
com. android. deskclock. AlarmKlaxon. play
com. android. deskclock. AlarmKlaxon. startAlarm
com. android. deskclock. AlarmKlaxon. stop
android. os. Vibrator. vibrate
android. media. MediaPlayer. start
android. media. MediaPlayer. stop
android. os. Vibrator. cancel

Figure 4. Part of the callgraph for onStartCommand() in AlarmKlaxon of DeskClock application. The entry point leads to four critical system APIs.

Through the above two cases, we can claim that our prototype of Diordna is able to detect permission re-delegation vulnerability and generate appropriate test case specifications, through there are some limitations as we discussed previously.

```
Intent {
  Extra : [ intent.extra.alarm -> {
    [ExtraType = ParcelableExtra],
    [TYPE = com.android.deskclock.Alarm],
    @ != null,
    .alert : (NOT (@ != null)),
    .silent : (NOT (@ != 0)),
    .vibrate : (NOT (@ == 0)),
  }];
}
```

(a)

```
Alarm alarm = new Alarm();
Alarm.silent = false;
Alarm.vibrate = true;
Intent intent = new Intent
("con.android.deskclock.ALARM_ALERT");
Intent.putExtra ("intent.extra.alarm", alarm);
startService (intent);
```

(b)

Figure 5. Test case specification for Alarm application (a) and a real test case guided by the specifications (c). They are for the system API: android.os.Vibrator.vibrate. Symbol '@' means the current object.

IV. CONCLUSION

As the proliferation of Android market and its applications, many recent studies have been done on the security problems of the applications. Unlike most researchers focus on suspicious operations such as leaking privacy, we pay attention to one of the applications' inner flaws which may lead to permission re-delegation. We present Diordna to detect such vulnerability in Android applications and to generate Intent oriented test case specifications. Diordna aims to help developers decrease the possibility of their applications being unintentional deputies.

We test Diordna on two system applications on Android 2.3.3 and the result shows that our solution can work well in certain cases. As a static tool, we cannot handle some specific features in both object oriented language and Android platform. This would be our future work to improve the capability of our tool. Possible improvements may include more precise callgraph and control flow graph construction and more complete method resolution. We also anticipate more researchers to do deeper studies in this area, to develop more effective detection and test case generation methods. A comprehensive test case generation rather than just generating for Intent object would be promising and can help other relevant researches.

Acknowledgement: The work has been supported by the National Natural Science Foundation of China under Grant No. 61170240 and 61070192.

REFERENCES

[1] Nielsen Company. 40 Percent of U.S. Mobile Users Own Smartphones; 40 Percent are Android.
http://blog.nielsen.com/nielsenwire/online_mobile/40-percent-of-u-s-mobile-users-own-smartphones-40-percent-are-android/.

[2] Nielsen Company. In U.S. Smartphone Market, Android is Top Operating System, Apple is Top Manufacturer.
http://blog.nielsen.com/nielsenwire/online_mobile/in-u-s-smartphone-market-android-is-top-operating-system-apple-is-top-manufacturer/.

[3] C. Gibler, J. Crussell, J. Erickson, and Chen. H. "AndroidLeaks: Detecting Privacy Leaks In Android Applications". Technical Report, UC Davis. http://www.cs.ucdavis.edu/research/tech-reports/2011/CSE-2011-10.pdf.

[4] W. Enck, P. Gilbert, B. G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. "TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones". In Proceedings of OSDI 2010. Oct., 2010. Vancouver, BC.

[5] Y. Zhou, X. Zhang, X. Jiang, and V. W. Freeh. "Taming Information-Stealing Smartphone Applications (on Android)". In Proceedings of the 4th International Conference on Trust and Trustworthy Computing (TRUST 2011). June, 2011. Pittsburgh, PA.

[6] M. Dietz, S. Shekhar, Y. Pisetsky, and A. Shu. "QUIRE: Lightweight Provenance for Smart Phone Operating Systems". In Proceedings of USENIX Security 2011. Aug., 2011. San Francisco, CA.

[7] T. Cannon. Android Data Stealing Vulnerability.
http://thomascannon.net/blog/2010/11/android-data-stealing-vulnerability/.

[8] X. Jiang. Smartphone security and privacy.
http://www.csc.ncsu.edu/faculty/jiang/.

[9] Y. Zhou, and X. Jiang. "An Analysis of the AnserverBot Trojan". Technical Report, NC State University.
http://www.csc.ncsu.edu/faculty/jiang/pubs/AnserverBot_Analysis.pdf.

[10] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri. "A Study of Android Application Security". In Proceedings of USENIX Security 2011. Aug., 2011. San Francisco, CA.

[11] E. Chin, A. P. Felt, K. Greenwood, and D. Wagner. "Analyzing Inter-Application Communication in Android". In Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services (MobiSys 2011). June 28–July 1, 2011. Bethesda, Maryland, USA.

[12] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. "Android Permission Demystified". In Proceedings of ACM CCS 2011. Oct. 17–21, 2011. Chicago, Illinois, USA.

[13] A. P. Felt, J. J. Wang, A. Moshchuk, S. Hanna, and E. Chin. "Permission Re-Delegation: Attacks and Defenses". In Proceedings of USENIX Security 2011. Aug., 2011. San Francisco, CA.

[14] M .Grace, Y. Zhou, Z. Wang, and X. Jiang. "Systematic Detection of Capability Leaks in Stock Android Smartphones". In Proceedings of NDSS 2012. Feb., 2012. San Diego, CA.

[15] Android Developer Reference. http://developer.android.com/

[16] pxb1988. dex2jar: A tool for converting Android's .dex format to Java's .class format. http://code.google.com/p/dex2jar/.

[17] IBM T.J. Watson Research Center. T.J. Watson Libraries for Analysis (WALA). http://wala.sourceforge.net/.

[18] IBM T.J. Watson Research Center. Shrike: Toolkits for Reading, Modifying, and Writing Java Bytecodes.
http://wala.sourceforge.net/wiki/index.php/Shrike_technical_overview.

[19] A. P. Felt. MediaPlaubackService should require WAKE_LOCK permission.
http://code.google.com/p/android/issues/detail?id=14660.

[20] A. P. Felt. DeskClock service should require WAKE_LOCK permission. http://code.google.com/p/android/issues/detail?id=14659.