MonkeyDroid: Detecting Unreasonable Privacy Leakages of Android Applications

Kai Ma[1(✉)], Mengyang Liu[1(✉)], Shanqing Guo[1(✉)], and Tao Ban[2(✉)]

[1] Department of Computer Science and Technology, Shandong University, Jinan, China
{makaisdu,mengyang.liu}@gmail.com, guoshanqing@sdu.edu.cn
[2] NICT, Tokyo, Japan
bantao@nict.go.jp

Abstract. Static and dynamic taint-analysis approaches have been developed to detect the processing of sensitive information. Unfortunately, faced with the result of analysis about operations of sensitive information, people have no idea of which operation is legitimate operation and which is stealthy malicious behavior. In this paper, we present Monkeydroid to pinpoint automatically whether the android application would leak sensitive information of users by distinguishing the reasonable and unreasonable operation of sensitive information on the basis of information provided by developer and market provider. We evaluated Monkeydroid over the top 500 apps on the Google play and experiments show that our tool can effectively distinguish malicious operations of sensitive information from legitimate ones.

Keywords: Android security · Privacy leakage detection · Static taint analysis · Natural language processing

1 Introduction

According to the statistical data from Strategy Analytics [1], Android operating system accounted for about 84.6 % of the mobile terminal market in 2014. Because of the importance of mobile devices in our daily life such as storing a large amounts of sensitive personal information and the openness of the Android platform, the android platform has become an ideal land of the wanton growth of malicious software. As more and more apps are used for private and privileged tasks, concerns are also rising about the consequences of failure to protect or respect users privacy.

As a result, many approaches have been proposed to automatically reveal privacy disclosures in Android apps and can be grouped into two major categories: static analysis (e.g..ScAndroid [2], Androidleaks [3], Chex [4]) and dynamic analysis(e.g..Taintdroid [5], Appsplayground [6]). Although these work could be successful to reveal the operations of sensitive information in the Android apps. However,there is no an approach to judge the legitimacy of the detected flows of sensitive information(e.g..location, device identifier).

S. Arik et al. (Eds.): ICONIP 2015, Part III, LNCS 9491, pp. 384–391, 2015.
DOI: 10.1007/978-3-319-26555-1_43

In this paper, we propose Monkeydroid, an approach to automatically pinpoint whether the android application will leak android users sensitive information in a mobile device. With the help of application description and Android API document, this approach leads to a significant reduction in false positives by distinguishing the reasonable and unreasonable behaviors of android application about sensitive information. We evaluated Monkeydroid over the top 500 apps on the Google play. Our experiments show that our tool can effectively distinguish malicious operations of sensitive information from legitimate ones.

The contributions of this paper are as follows:

- We propose an approach to judge the legitimacy of operations of sensitive information in Android applications based on natural language processing of application description and static taint analysis.
- We implement a system named Monkeydroid, which is a prototype of our approach to detect privacy leakages of Android applications. We evaluate Monkeydroid on the top 500 apps from the Google Play store. The results show Monkeydroid has a high accuracy and high performance in distinguishing malicious operations of sensitive information from legitimate ones.

The rest of this paper is organized as following. In Sect. 2, we introduce the problem statement of our work. Section 3 gives the overview of Monkeydroid and implementation of Monkeydroid. In Sect. 4, we evaluate Monkeydroid. Section 5 presents the limitation of Monkeydroid. We draw the conclusions in Sect. 6.

2 Problem Statement

Let us consider a weather application (com.devexpert.weather.apk) that the users location need to be sent to the server when the user utilizes the app to obtain the local weather information. Using current detection systems to analyze the app, we will get a report which contains a privacy disclosure about users location, regardless of the fact that the apps legitimate function depends on users location. In fact,we lack an effective tool to inform the user which are legitimate and malicious privacy disclosures reported by static or dynamic analysis.

Based on the above facts, we propose one solution to this problem, where we try to use application description to judge the legitimacy of operations of sensitive information in an application.

Question 1. Why do we try to use market description to help us analyze the privacy disclosure of Android apps?

The programmer of the application utilizes code to realize his intention in the app and makes use of natural language to express his intention in the application description if he is honest. If the intention of code can match the intention of application description, we think that the user can know how the apps deal with his privacy by reading the application description and we can believe that privacy disclosure does not exist in the app. If the application description does

not contain some kinds of sensitive information but the code of the application contains them, we can identify benign and malicious operations of sensitive information through our approach.

Question 2. How to bridge the semantic gap between apk and its description?

Although users can easily understand the meaning of a natural language document, it's difficult for computers to perform like human beings to understand the meaning of natural language sentences. Therefore, an effective intermediate expression is in need to help the computer automatically breaks the semantic gap between application and its description. Considering that the meaning of a well constructed natural language sentence that describes operations can be regarded as a statement, it is reasonable to convert these sentences to logic expressions. Recent research has shown the adequacy of using First-Order-Logic expression for NLP(Natural Language Processing) related analysis tasks [7]. So we construct our intermediate-representation, semantic graph, based on the First-Order-Logic expression.

3 Monkeydroid

In this section, we present the architecture of Monkeydroid (in Fig. 1) and give the details of design and implementation.

3.1 Extracting Sensitive-Information Behaviors

Monkeydroid unzips an apk file and decomposes the Dalvik bytecode executable file into Jimple representation, which is a typed-3 address intermediate representation suitable for analysis and optimization on the Soot framework. Leveraging layout files and manifest file, Monkeydroid constructs precise call graph and ICFG(Inter-procedural Control-Flow Graph) [8]. Monkeydroid marks sensitive sources and outgoing channels(sinks) provided by SuSi [10] and makes use of static taint analysis to reveal flows from sources to outgoing channels(sinks).

After static taint analysis, Monkeydroid reduced sensitive information flows to source-to-sink form. Then, Monkeydroid constructs semantic graphs(in Fig. 3)

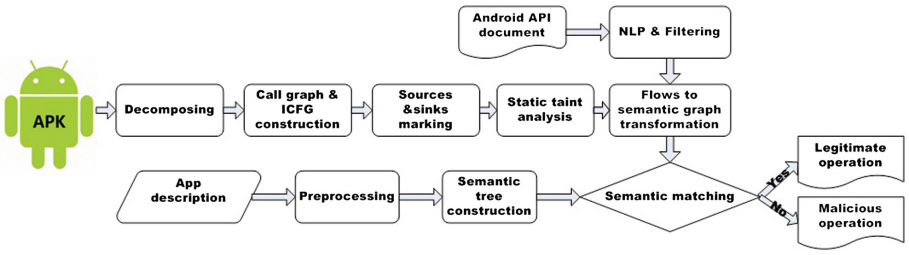

Fig. 1. The architecture of Monkeydroid

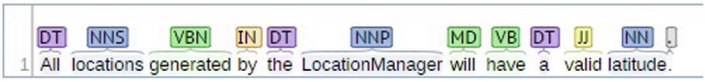

Fig. 2. Android API document

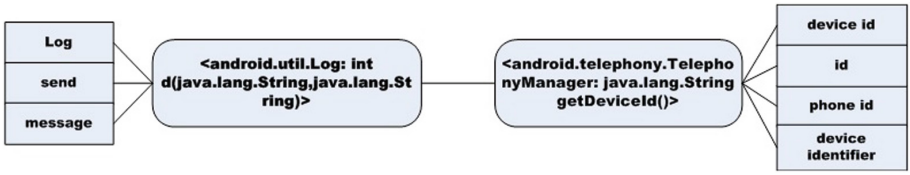

Fig. 3. The semantic graph of this flow

of the simplified flows on the basis of knowledge base. The knowledge base is defined as a collection of verbs or nouns associated with particular API name. Knowledge atoms can be represented as <API name - verb or noun set>. To ensure the accuracy of the semantic graph, Monkeydroid uses Android API document to generate a knowledge base containing semantic information of sensitive resource related API. In the Android API document, each API name (e.g. public double getLatitude()) is followed by several nature language sentences (e.g. "Get the latitude, in degrees.") describing the function of this API. For the remaining sensitive API, Monkeydroid uses the Stanford parser to identify the part of speech (POS, such as nouns and verbs) a particular word in a sentence in the description of an API (in Fig. 2). After POS tagging, Monkeydroid traverses the tagged sentences and extracts the nouns or verbs respectively for API in the source or the sink list. The extracted verbs or nouns are organized in the form of knowledge atoms mentioned above.

3.2 Application Description Analysis

This subsection presents the approach of the transformation between application description and corresponding semantic tree used in Monkeydroid. This part in Monkeydroid is implemented as the following two units:

1. Preprocessor. The preprocessor is designed for segmenting target application description. It accepts the natural language description of target application and generates a list of separated sentences which contains the keyword of the source generated by the program analyser. The separated sentences are put into an automation to check if they contains a keyword.
2. Semantic-tree construction. The semantic-tree construction is designed for the generation of a semantic-tree of a sentence which contains the keyword of the source. It accepts the sentence list generated by the preprocessor and provide a semantic-tree for the checker. The semantic-tree is represented by the FOL representation. We implement the semantic-tree construction with the help of Stanford NLP tools [9].

3.3 Semantic Graphs of APK and Application Description Matching

This subsection is about to find the contradiction between program context and application description. After the construction of the semantic graph and the semantic tree, Monkeydroid tries to justify the rationality of the sensitive behavior found by our framework. A semantic graph is defined as a graph containing a verb set and a noun set, all of which is related to the result of the static taint analysis. A semantic tree is defined as a tree structure that is essentially a First-Order-Logic expression. In the matching process, Monkeydroid traverses all of the leaf nodes. The leaf node that contains noun in the semantic graph would yield a procedure to check its ancestor nodes. If there is a ancestor node containing a verb in the corresponding semantic graph, the semantic graph matches the semantic tree, otherwise the semantic graph does not match the semantic tree. The algorithm(in Algorithm 1) shows the process in pseudo-code.

```
   Input: SemGraph, SemTree
   Output: inDesc
 1 inDesc = False;
 2 for node in SemTree.leafNodes do
 3 |   if node.content in SemGraph.nounSet then
 4 |   |   iter = node;
 5 |   |   while iter.hasParent do
 6 |   |   |   iter = iter.parent;
 7 |   |   |   if iter.content in SemGraph.verbSet then
 8 |   |   |   |   inDesc = True;
 9 |   |   |   |   return inDesc;
10 |   |   |   end
11 |   |   end
12 |   end
13 end
14 return inDesc;
```

Algorithm 1. Match semantic graph with semantic tree

4 Evaluation

4.1 Study Subjects

In the experiment, we evaluate the effectiveness of Monkeydroid over the top 500 apps downloaded from the Google play store during March, 2014. Although there are more than one million applications on the Google play, most smartphone users only concentrate on the top applications and it's rational for us to choose the top 500 apps as the study subjects. By testing these apps with Monkeydroid, we evaluate whether Monkeydroid can precisely identify benign operation on sensitive information, particularly those delivering sensitive information to the network.

4.2 Experimental Environment

We setup our evaluation on a sever with 12×2.00 GHz processors and 32.0 GB physical memory. Each analysis task (for analyzing an app) is assigned to 8×2.00 GHz processors and 16 GB, which runs JDK 1.7.0 21. Because of either insufficient memory or failure of type resolving, Monkeydroid cannot analyze some apps in the evaluation and we share same problem with other flowDroid-dependent tools [8]. Thus, below we show our experimental results over the remaining apps.

4.3 Results

We analyze the top 500 apps, among which 164 apps fail to go through and 256 apps contain behaviors of the sensitive information(e.g..location, IMEI, contact). Monkeydroid identifies that behaviors of the sensitive information are legitimate in 65 apps and unreasonable in 227 apps by matching apk with app description.

We show a part of the summarized result in Table 1 for these apps. From left to right of the table, the table shows apps package name, sensitive information and sink. If the sensitive information is sent to the outside through the sink, the flow could be recorded in the table. The fourth column of the table is "In description" and we fill "Yes" or "No" in the blank if the sensitive information mentioned in the market description or not. If the operation described in the market description, we take it as the rational operation and use a dot to mark in the table. To check the result of Monkeydroid, we also utilize the VirusTotal to test the apps and record how many anti-virus engines in VirusTotal identify the app as malicious.

From this table, our observations and findings are as follows. First, we can see that the most frequently leaked sensitive information is device ID (IMEI). The related research work such as Taintdroid [5] can verify this result. Second, if the sensitive information mentioned in the market description, this operation of app is likely to be rational. Third, the applications developed by the mature company like Facebook, Twitter are usual to be reinforced and we can't analyze these apps completely.

5 Limitation

There are several limitations with Monkeydroid. As other static analysis systems on Android, Monkeydroid cannot detect the disclosures caused by Java reflection, code encryption, JNI calls, or dynamical code loading. As mentioned in Sect. 3.1, knowledge base only from Android API is not complete and this can produce false positives in matching semantic graphs. Faced with multiple sentences about one kind of sensitive information, Monkeydroid cannot take accurate analysis and this is the major problem we will solve in the future work.

Table 1. Partial experimental results

Package name	Sensitive information	Sink	In description	Rational	Not rational	Virus total
Vitamio Plugin.apk	IMEI	Log	Yes	–	•	0/57
	IMSI	Log	No	–	•	
	SimSerialNumber	Log	No	–	•	
	SimCountryIso	Log	No	–	•	
com.whatsapp.wallpaper.apk	location	Internet	No	–	•	0/57
com.gameloft.android.apk	location	Internet	No	–	•	0/57
com.photo.ghost.prank.apk	location	Internet	No	–	•	0/57
AccuWeather 3.2.14.1	IMEI	Log	No	–	•	0/57
	IMEI	Internet	No	–	•	
	location	Log	No	–	•	
	location	Internet	Yes	•	–	
Amazon 2.9.7	location	Internet	No	–	•	0/57
Backgrounds HD Wallpapers 2.0.1	IMEI	Log	No	–	•	0/57
	IMEI	Internet	No	–	•	
	Contact	Log	Yes	•	–	
Chase Mobile 3.16	Contact	Internet	Yes	•	–	0/57
ebay 2.6.0.98	Location	Internet	Yes	•	–	0/57
HotPads 3.1	Location	Internet	Yes	•	–	0/55

6 Conclusion

In this paper, we propose an approach to judge the legitimacy of operations of sensitive information in Android Applications based on natural language processing of application description and static taint analysis. We implement a system named Monkeydroid, which is a prototype of our approach. Compared with previous work, Monkeydroid effectively rules out the legitimate and malicious privacy disclosures, which exposes those privacy leakages that cannot be associated with apps functions. As a result, Monkeydroid can greatly increase the detection rate of threatening privacy leaks, and at the same time, considerably prompt developer to pay attention to the description of apps and to be honest.

Acknowledgments. This work is partially supported by National Natural Science Foundation of China (61173068, 61173139), Program for New Century Excellent Talents in University of the Ministry of Education, the Key Science Technology Project of Shandong Province (2014GGD01063), the Independent Innovation Foundation of Shandong Province(2014CGZH1106) and the Shandong Provincial Natural Science Foundation (ZR2014FM020, ZR2014FM031).

References

1. https://www.strategyanalytics.com/access-services/devices/mobile-phones
2. Fuchs, A.P., Chaudhuri, A., Foster, J.S.: Scandroid: automated security certification of android applications. Manuscript, University of Maryland **2**(3), (2009). http://www.cs.umd.edu/avik/projects/scandroidascaa

3. Gibler, C., Crussell, J., Erickson, J., Chen, H.: AndroidLeaks: automatically detecting potential privacy leaks in android applications on a large scale. In: Katzenbeisser, S., Weippl, E., Camp, L.J., Volkamer, M., Reiter, M., Zhang, X. (eds.) Trust 2012. LNCS, vol. 7344, pp. 291–307. Springer, Heidelberg (2012)
4. Lu, L., Li, Z., Wu, Z., et al.: Chex: statically vetting android apps for component hijacking vulnerabilities. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security, pp. 229–240. ACM (2012)
5. Enck, W., Gilbert, P., Han, S., et al.: TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. ACM Trans. Comput. Syst. (TOCS) **32**(2), 5 (2014)
6. Rastogi, V., Chen, Y., Enck, W.: AppsPlayground: automatic security analysis of smartphone applications. In: Proceedings of the third ACM Conference on Data and Application Security and Privacy, pp. 209–220. ACM (2013)
7. Pandita, R., Xiao, X., Zhong, H., et al.: Inferring method specifications from natural language API descriptions. In: Proceedings of the 34th International Conference on Software Engineering, pp. 815–825. IEEE Press (2012)
8. Arzt, S., Rasthofer, S., Fritz, C., et al.: Flowdroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. ACM SIGPLAN Not. **49**(6), 259–269 (2014). ACM
9. Pandita, R., Xiao, X., Yang, W., et al.: WHYPER: towards automating risk assessment of mobile applications. In: USENIX Security 2013 (2013)
10. Rasthofer, S., Arzt, S., Bodden, E.: A machine-learning approach for classifying and categorizing android sources and sinks. In: 2014 Network and Distributed System Security Symposium (NDSS), February 2014, to appear. http://www.bodden.de/pubs/rab14classifying.pdf