Poster: Android Inter-Component Communication Analysis with Intent Revision

Cong Tian, Congli Xia, and Zhenhua Duan
ICTT and ISN Lab, Xidian University, Xian 710071, P.R. China
ctian@mail.xidian.edu.cn,zhhduan@mail.xidian.edu.cn

1 INTRODUCTION

Android has been one of the most popular platforms for smart phones, which has reached a 81.2% share in the mobile-phone market. With smart phones being ubiquitous, hackers are most likely to attack them to catch privacy of users. Android applications (also called Android apps) have been proved the effective target.Google Play store has provided billions of Android apps, but unfortunately, the advance has a dark side because security cannot be ensured by many Android apps. Hence, more and more attention has been paid to Android malware. Taint flow analysis has been proved an effective approach to providing potential malicious data flows. It aims at determining whether a sensitive data flows from a *source* to a *sink*. The analysis can be executed either dynamically or statically. Dynamic taint analysis [5] relies on testing to reach a appropriate code coverage criterion. It is able to precisely pinpoint leaks, but may be incomplete in exploring all possible executing paths. In contrast, static analysis takes all the possible paths for consideration. But most of the static analyses available for Android apps [1, 3] are inner-component based analysis which are unable to detect leaks across-components.

Even though most of the privacy leaks happen in a single component, lots of inter-components privacy leaks have been reported. Thus, inner-component taint analysis is not enough to detect leaks. Efforts have also been devoted to implement static analysis for Android [2]to supply us with a relatively satisfactory outcome. Among them, Inter-Component Communication (ICC) [4] analysis plays important roles since ICC values can facilitate a precise consequent.

However, the current ICC analyses only consider ICC links between components where reuse and revision of an Intent across-component are not considered. Thus, lots of potential leaks will escape from being tracked in the succeeding ICC leak detection. With this motivation, in this paper, we devote to ICC analysis on reused and revised Intents. To do so, first, ICC values are analyzed by taking reused and revised Intents into account. With this basis, target components of Intents are analyzed and ICC Graphs are built. On an ICCG, all the ICC flows, which are useful in tracking leaks across-components, are contained. This will lay a critical foundation to the succeeding taint flow analysis. The proposed approach has been implemented in a tool called ICC-Analyzer (ICCA) where IC3 is integrated in for providing ICC values of the Intents which are not reused or revised.

ICSE '18 Companion, May 27-June 3, 2018, Gothenburg, Sweden
© 2018 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-5663-3/18/05.
https://doi.org/10.1145/3183440.3194957

2 APPROACH OVERVIEW

Our approach is to analyse ICC values of Intents including the *reused* and *revised* ones. For clarity, first, we define *Reused Intents* and *Revised Intents*: (1) An Intent is called a reused Intent if it is acquired by the ICC method `getIntent()` outside the component where it is created; (2) Whenever a reused Intent is modified, it is called a revised Intent. The phenomenon that an Intent is reused across-component is called *Intent Reuse* (IR) and the circumstance that a reused Intent is revised across-component is denoted as *Inter-Component Intent Revision* (ICIR). To characterize communications between components, an ICC link $l : m \rightarrow C$ has been defined to link two components with an Intent. Here m is an ICC method in the source component and C is the target component. In the target component, after being caught by method `getIntent()`, it is possible that the Intent is revised, which cannot be tracked by the existing ICC leak detecting tools, such that we can exploit it to start another component.

The framework of our approach is shown in Figure 1. Our aim is to analyse ICC values of Intents including the *reused* and *revised* ones, and construct the ICC graphs eventually. As known to all, an Android app runs on Dalvik virtual machine. Thus, we first convert the Dalvik bytecode of an app to an intermediate representation namely Jimple with Soot . Then, with the obtained Jimple files and the bytecode of the app, IC3 is employed to infer the involved ICC methods and ICC values of Intents excluding the *reused* and *revised* ones. Further, based on the acquired information, the Jimple programs are instrumented such that ICC values of *reused* and *revised Intents* are able to be precisely acquired by reusing IC3. When all the ICC values are obtained, target components of Intents can be inferred subsequently. Finally, ICCG are constructed where a complete set of ICC flows are contained.

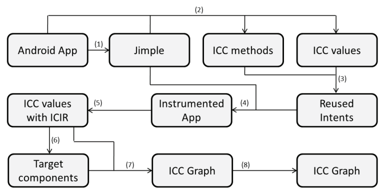

Figure 1: Approach overview

To acquire ICC values of *reused* and *revised Intents*, Jimple code of the app is instrumented such that IC3 can be reused for achieving our goal. In Android apps, method `getIntent()` is utilized to reuse an Intent which already occurs in another component. An Intent as well as all its reuses share the same memory block physically. Thus, they cannot be analyzed individually. Otherwise, the obtained ICC values will be inaccurate. To acquire accurate ICC values of *reused* and *revised Intents* with the help of IC3, in the component where the Intent is reused, an extra static field is added to create a new Intent. Afterwards, we should promote it succeed in making all the *reused Intents* point to the newly created Intent. Even though instrumenting Jimple in this way will change the semantics of the original program, it is helpful in making IC3 be able to acquire the modified ICC values of the *reused Intent*. To do that, for each newly created Intent `i`, we first acquire the set of components C where Intent `i` is reused or revised. Then, each of the component $c \in C$ is instrumented by:

(1) Generating a new static field by inserting "`static Intent myIntent=new Intent()`" inside the component; and
(2) replacing "`getIntent()`" with "`myIntent`".

We just use "`myIntent`" to indicate the newly introduced Intent.

3 IMPLEMENTATION AND EVALUATION

We have implemented our approach in a tool named ICCA to analyze ICC values with ICIR and construct ICCGs of Android apps for the convenience of the succeeding ICC leak detection. The evaluation of our approach addresses the following two research questions: (1) How does ICCA perform when analyze ICC values with ICIR? and (2) As an ICC analysis tool for Android apps, how ICCA can precisely match the targets of Intents?

3.1 ICC Analysis with ICIR

By experiments, we can obtain that 37 and 36 *revised Intents* exist in *GooglePlay* and *MalGenome*, respectively. We apply ICCA in analyzing ICC values of the 73 *revised Intents*. Table 1 illustrates a bird's eye view of the whole experiment. The l.h.s of Table 1 shows the seven attributes **A**ction, **C**ategory, **T**ype, **D**ata, **F**lag, **E**xtra, and **Com**ponent of the 37 different *revised Intents* in *GooglePlay*. The r.h.s illustrates the attributes of the 36 *revise Intents* in *MalGenome*. Note that in the table, '–' means that the relative value is captured but not modified, and '√' indicates that the revised value is successfully acquired. As shown in Table 1, ICC values of all the 73 *revised Intents* are successfully captured which are unable to be obtained by all the existing ICC analysis tools.

3.2 Matching Target Components

In this part, we illustrate the target components of different Intents in *GooglePlay* and *MalGenome* matched by ICCA. The results are compared with the target components matched with IC3. All the Intents are classified into three categories: explicit, implicit and reused ones. We record the numbers of Intents in different categories and numbers of the matched target components.

The results on *GooglePlay* and *MalGenome* are presented in Table 2. The first column are the sets of data; the second one shows the categories of Intents; the third one illustrates the numbers of the involved Intents in the relative category. The right-most two columns present the numbers of the target components identified by IC3 and ICCA, respectively. As shown in the experiment, both IC3 and ICCA can identify most of the explicitly defined target components of Intents (94.8% in *GooglePlay* and 98.7% in *MalGenome*). For implicit ones, a small part (1.3% in *GooglePlay* and 0.7% in *MalGenome*) of them are acquired by ICCA whereas null of them can be obtained by IC3. The success rate of ICCA is low since implicit Intents are frequently used to launch target components in other apps which

Table 1: ICC-Values of ICIR in *GooglePlay* and *MalGenome*

GooglePlay								*MalGenome*							
I	A	C	T	D	F	E	Com	I	A	C	T	D	F	E	Com
1	–	–	–	–	–	√	√	1	–	–	–	–	–	–	√
2	–	–	–	–	–	√	√	2	–	–	–	–	–	–	√
3	–	–	–	–	–	√	√	3	–	–	–	–	–	–	√
4	–	–	–	–	–	√	√	4	–	–	–	–	–	–	√
5	–	–	–	–	–	√	√	5	–	–	–	–	–	–	√
6	–	–	–	–	–	√	√	6	–	–	–	–	–	–	√
7	–	–	–	–	–	√	√	7	–	–	–	–	–	√	√
8	–	–	–	–	–	√	√	8	–	–	–	–	–	–	√
9	–	–	–	–	–	√	√	9	–	–	–	–	–	–	√
10	–	–	–	–	–	√	√	10	–	–	–	–	–	√	√
11	–	–	–	–	–	√	√	11	–	–	–	–	√	√	√
12	–	–	–	–	–	√	√	12	–	–	–	–	√	–	√
13	–	–	–	–	–	√	√	13	–	–	–	–	√	√	√
14	–	–	–	–	–	√	√	14	–	–	–	–	√	√	√
15	–	–	–	–	–	√	√	15	–	–	–	–	√	√	√
16	–	–	–	–	–	√	√	16	–	–	–	–	√	√	√
17	–	–	–	–	–	√	√	17	–	–	–	–	√	–	√
18	–	–	–	–	–	√	√	18	–	–	–	–	√	√	√
19	–	–	–	–	√	–	–	19	–	–	–	–	√	√	√
20	–	–	–	–	√	–	–	20	–	–	–	–	–	√	√
21	–	–	–	–	√	√	–	21	–	–	–	–	–	–	√
22	–	–	–	–	√	–	–	22	–	–	–	–	–	–	√
23	–	–	–	–	√	–	–	23	–	–	–	–	–	–	√
24	–	–	–	–	–	√	–	24	–	–	–	–	–	–	√
25	–	–	–	–	√	–	–	25	–	–	–	–	–	–	√
26	–	–	–	–	√	–	–	26	–	–	–	–	–	–	√
27	–	–	–	–	√	–	–	27	–	–	–	–	–	–	√
28	–	–	–	–	√	–	–	28	–	–	–	–	–	–	√
29	–	–	–	–	√	–	–	29	–	–	–	–	–	–	√
30	–	–	–	–	√	√	–	30	–	–	–	–	–	–	√
31	–	–	–	–	–	–	√	31	–	–	–	–	–	–	√
32	–	–	–	–	–	√	√	32	–	–	–	–	–	–	√
33	–	–	–	–	–	√	√	33	–	–	–	–	–	–	√
34	–	–	–	–	–	–	√	34	–	–	–	–	–	–	√
35	–	–	–	–	√	–	–	35	–	–	–	–	–	–	√
36	–	–	–	–	√	–	–	36	–	–	–	–	–	–	√
37	√	–	–	–	–	–	√	——							

Table 2: Target components

Datasets	Intents Categories	Numbers	Matched Targets	
			IC3	ICCA
GooglePlay	explicit	14906	14134	14134
	implicit	26288	0	338
	reuse	51	0	49
MalGenome	explicit	10440	10307	10307
	implicit	13187	0	93
	reuse	41	0	41

cannot be acquired without the runtime environment. Thus, our result is reasonable. For the reused Intents, ICCA can acquire almost all the target components while null of them are acquired by IC3. In this experiment, we compare the results of ICCA in ICC analysis only with the newest ICC analysis tool IC3 as it is an improvement of Epicc. To be best of our knowledge, IC3 and Epicc are the only ICC analysis tools publicly available.

ACKNOWLEDGEMENT

This research is supported by the NSFC under grant No. 61420106004, 61751207 and 61732013. Cong Tian and Zhenhua Duan are the corresponding authors.

REFERENCES

[1] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android permissions demystified. In *Proceedings of the ACM Conf. on Computer and Communications Security (CCS)*, 2011.
[2] A. P. Fuchs, A. Chaudhuri, and J. Foster. Scandroid: Automated security certification of android applications. *University of Maryland, Tech. Rep. CS-TR-4991*, 2009.
[3] M. Grace, Y. Zhou, Z. Wang, and X. Jiang. Systematic detection of capability leaks in stock android smartphones. *NDSS '12*, 2012.
[4] L. Li, A. Bartel, T. F. Bissyande, J. Klein, Y. L. Traon, S. Arzt, S. Rasthofer, E. Bodden, D. Octeau, and P. Mcdaniel. Iccta: Detecting inter-component privacy leaks in android apps. *ICSE*, 2015.
[5] R. Xu, H. Saidi, and R. Anderson. Aurasium: practical policy enforcement for android applications. *USENIX Security 2012*, pages 27–27, Berkeley, USA, 2012.