2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)

FastDroid: Efficient Taint Analysis for Android Applications

Jie Zhang, Cong Tian* and Zhenhua Duan*
ICTT and ISN Lab, Xidian University, Xi'an 710071, P.R. China
Emails: zhangjie@xidian.edu.cn, ctian@mail.xidian.edu.cn, zhhduan@mail.xidian.edu.cn

***Abstract*—In recent years, sensitive data leaks of Android system attracted significant attention. The traditional tools for detecting leaks usually focus on the precision and recall with few of them addressing the importance of the efficiency. The high costs of these tools often make them fail in analyzing apps in large scale and thus block them from wide usage in practice. In this paper, we propose FastDroid, an efficient and precise tool for detecting sensitive data leaks in Android apps. First, a flow-insensitive taint analysis is conducted to construct the taint value graph (TVG) which is defined to describe the process of taint propagation. Then, potential taint flows (PTFs) are extracted from TVG. Finally, the PTFs are checked on the control flow graph (CFG) to acquire the real taint flows. FastDroid is evaluated on three test suites. The results show that FastDroid maintains a high precision and recall; meanwhile it improves the efficiency significantly.**

***Keywords*-Android; security; static analysis; taint analysis;**

I. Introduction

Android has over two billion monthly active users and dominates the smartphone market. However, the popularity also arouses the attention of the attackers who try to steal the sensitive data of users. Although Google constantly introduces new security measures to Android system, the leak of sensitive data is still a severe problem. Thus, the security of Android apps attracts significant attention, especially the area on sensitive data leaks.

Taint analysis aims to prevent sensitive data leaking by detecting a taint flow which starts from a Source (an API method returning sensitive data) to a Sink (an API method leaking data out). The traditional approaches for static taint analysis, such as FlowDroid [1], are based on the framework of data flow analysis. As an example, FlowDroid has a high precision and recall ratio because of its comprehensive and accurate modeling of Android runtime. However, we found that FlowDroid costs a lot of time in our experiment, especially for large scale Android apps. In our view, FlowDroid utilizes a precise inter-procedural dataflow analysis framework called IFDS which is designed for general purposes. The cost of the framework could be high to solve specific problems, such as taint analysis. Besides, it is quite expensive to combine the forward and backward searching in the taint analysis of FlowDroid especially when a taint value has numbers of aliases.

To improve the efficiency, we propose FastDroid, a novel static taint analysis tool that supports efficient taint analysis of apps in large scale, and meanwhile maintains high precision and recall. FastDroid utilizes a different strategy for taint analysis that focuses on exploring the propagation of taint values, rather than using a data flow analysis. Specifically, our taint analysis is conducted in two steps: first, a preliminary flow-insensitive taint analysis is implemented to create a taint value graph (TVG) which is a graph we defined to describe the relation among taint values; second, potential taint flows (PTFs) are extracted from the TVG and checked on the control flow graph (CFG) to eliminate the spurious taint flows. To improve the efficiency, all the taint values and their relevant statements are located first, then we attain feasible paths through them on the CFG. Given that the CFG is not considered in the first step, FastDroid does not require as much time as other tools. Moreover, the search space of statements to analyze in the second step is reduced and thus the efficiency is also improved.

We have evaluated FastDroid on a series of experiments. The result shows that FastDroid has a better performance on precision and recall and has a clearly smaller cost on time compared with the state-of-the-art tool FlowDroid 2.0.

The contributions of this paper are summarized as follows:

- We design TVG to describe the propagation of taint values, from which PTFs are extracted.
- We propose a novel method for taint analysis, which significantly increases the efficiency and maintains a high precision and recall.
- We develop FastDroid, a tool for taint analysis of Android apps, and evaluate it by several experiments.

II. Taint Value Graph and Potential Taint Flows

Generally, a taint flow contains a group of taint values that could be local or static variables, or fields of objects. Through taint values, sensitive data can be propagated in several ways such as assignment, parameter passing and alias. Once a taint value is leaked out, a taint flow is determined. To describe the taint propagation, we define TVG to express all the relations among taint values in an app. TVG is a directed graph with three features: 1) there exist one or more vertexes namely SC with an in-degree being 0; 2) each vertex has at least one path from the vertex SC to it; 3) there is no loop in the graph. A vertex represents a taint value and a directed edge represents a taint propagation, i.e. the source vertex taints the target vertex. The vertexes with 0 in-degree denote the sources.

2574-1934/19/$31.00 ©2019 IEEE
DOI 10.1109/ICSE-Companion.2019.00092

PTFs are defined as approximate estimations of real taint flows, which are extracted from the TVG. In our approach, a PTF is expressed by an ordered set of several ordered pairs. A taint propagation is formalized as a relation between two taint values in the ordered pairs. The sequence of the taint values in a taint propagation is formalized as the sequence of the ordered pairs in the ordered set. One PTF indicates a taint flow. Formally, An PTF PTF_i is defined below:

$$PTF_i = \{< SC_i, X_1 >, < X_1, X_2 >, \cdots, < X_n, SK_i >\}$$

where SC_i is a source, SK_i is a sink, $X_i (1 \leq i \leq n)$ is a taint value. The ordered pair $< X_{i-1}, X_i >$ denotes a relation of taint propagation, which means the latter is tainted by the former. The order pairs $< SC_i, X_1 >$ and $< X_n, SK_i >$ respectively denote that X_1 is tainted by a source SC_i and that X_n is leaked out by a sink SK_i.

III. Taint Analysis

The taint analysis of FastDroid consists of two steps. First, FastDroid constructs the TVG. We abstract the app as a tree structure in which all the leaf nodes are statements and design a special algorithm to process every leaf nodes of the tree with groups of taint rules. The taint rule indicates the relation of taint propagation between a new and known taint value. For example, a variable can be tainted by a known taint value through assignment. The algorithm applies the taint rules on each statement to judge whether it generates a new taint value. The taint values are included into the vertex set of TVG and their relations form the edges of TVG. After the algorithm ends, the TVG is constructed. The running time of the algorithm is polynomial in the amount of statements.

Second, FastDroid creates and checks the PTFs. Upon the TVG constructed, all PTFs can be extracted by searching the paths along the edges of TVG from sources to sinks. Because the PTFs probably involve in spurious taint flows due to the imprecision of flow-insensitive analysis, all the PTFs are checked on the CFG to acquire the real taint flows. The process for checking a PTF is as follows: a depth-first traversal is applied on the CFG to search any execution path that starts from the start point, passes through all the PTF relevant statements sequentially, and ends at the sink. If the path exists, the PTF is proved to be a real taint flow.

IV. Evaluation

FastDroid utilizes the CFG constructed by FlowDroid and is configured with sources and sinks imported from FlowDroid. To evaluate FastDroid, a series of experiments are conducted on three test suites: 1) DroidBench, a benchmark for taint analysis used in FlowDroid, 2) MalGenome that contains 1260 Android malware samples [2], and 3) 300 apps downloaded from Google-Play store. We compare the effectiveness and efficiency with FlowDroid 2.0.

Effectiveness: For DroidBench, FlowDroid has a precision 84.7% and recall 73.5%. By contrast, FastDroid achieves both a better precision of 91.0% and a better recall of 80.5%. The F-measure of FastDroid has a higher value of 0.85. For MalGenome, FastDroid detects 1143 apps containing at least one taint flow from the total 1260 malwares, while FlowDroid detects 1055 apps. For apps in Google-Play, FastDroid detects 220 apps containing taint flows out of 300 apps, while FlowDroid detects 159 apps.

Efficiency: We compare the runtime costed on taint analysis of the two tools. As shown in Fig. 1, both the tools are efficient, since the apps in DroidBench are all in small size. For MalGenome, the difference of runtime becomes more obvious. For Google-Play apps, with the growth of the size of apps, the runtime of both tools increases, however FastDroid increases more slowly than FlowDroid and costs much less time. To sum up, FastDroid is much more efficient than FlowDroid, especially on the large scale apps.

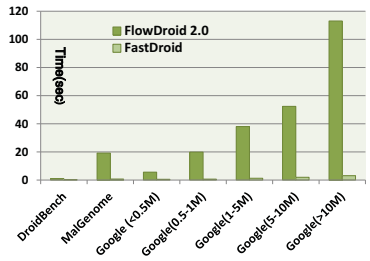

Figure 1. The comparison of the runtime of FlowDroid 2.0 and FastDroid on DroidBench, MalGenome and Google-Play in different size.

V. Conclusion

This paper presents FastDroid, a novel efficient and precise tool for taint analysis. Unlike the previous approaches, we focus on the propagation of taint values rather than the traditional data flow analysis to improve the efficiency. Our experiments show that FastDroid reaches a better performance on precision and recall and improves the efficiency significantly compared with FlowDroid.

Acknowledgment

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

References

[1] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. L. Traon, D. Octeau, and P. D. Mcdaniel, "Flowdroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps," *Programming Language Design and Implementation*, vol. 49, no. 6, pp. 259–269, 2014.

[2] Y. Zhou and X. Jiang, "Dissecting android malware: Characterization and evolution," *IEEE Symposium on Security and Privacy*, pp. 95–109, 2012.