POSTER: Detecting Inter-App Information Leakage Paths

Shweta Bhandari
MNIT Jaipur
Rajasthan, India
2014rcp9508@mnit.ac.in

Frederic Herbreteau
LaBRI - University of
Bordeaux, CNRS, France
fh@labri.fr

Vijay Laxmi
MNIT Jaipur
Rajasthan, India
vlaxmi@mnit.ac.in

Akka Zemmari
LaBRI - University of
Bordeaux, CNRS
zemmari@labri.fr

Partha S. Roop
The University of Auckland
Auckland, New Zealand
p.roop@auckland.ac.nz

Manoj Singh Gaur
MNIT Jaipur
Rajasthan, India
gaurms@mnit.ac.in

ABSTRACT

Sensitive (private) information can escape from one app to another using one of the multiple communication methods provided by Android for inter-app communication. This leakage can be malicious. In such a scenario, individual benign app, in collusion with other conspiring apps, if present, can leak the private information. In this work in progress, we present, a new model-checking based approach for inter-app collusion detection. The proposed technique takes into account simultaneous analysis of multiple apps. We are able to identify any set of conspiring apps involved in the collusion. To evaluate the efficacy of our tool, we developed Android apps that exhibit collusion through inter-app communication. Eight demonstrative sets of apps have been contributed to widely used test dataset named DroidBench. Our experiments show that proposed technique can accurately detect the presence/absence of collusion among apps. To the best of our knowledge, our proposal has improved detection capability than other techniques.

1. MOTIVATION

In Android, standard communication channels are based on Intent-based ICC. A recent study [9] showed that almost 85% of all apps in the market place perform inter-app communication via either explicit (11.3%) or implicit (73.1%) Intents. Unfortunately, the ICC model can be exploited by malware writers to deploy successful *Privilege escalation* attack [8] or *Collusion* attacks [7, 13]. Collusion refers to the scenario where two or more conspiring apps with a limited set of permissions communicate with each other to gain indirect privilege escalation and can perform unauthorized actions. In particular, an app with necessary permissions can access some sensitive information, send it using intents to another app, and this app can send the information out. This results in an information leak. The risk of this threat is that the individual app appears benign, but it may create privacy leakage path in the presence of another app(s).

Many techniques have been proposed for the analysis of Android apps. These include machine learning techniques and behavioral analysis. However, most of the existing works are focus on single app analysis [6, 10]. The attacker manages to plant the privacy leakage path by placing the source in one app and sink in another. Hence, can easily bypass detection by existing tools. Therefore to detect collusion, a set of apps need to be considered for analysis.

To demonstrate, we develop a scenario illustrated in Figure 1. It shows three colluding apps (named *DeviceId*, *DeviceId_Service* and *Collector*) that involve in the exfiltration of `device id`. All the three apps communicate via Intent objects. In the example, *DeviceId* invokes *DeviceId_Service* through `startService()` API call. Upon invocation, *DeviceId_Service* access sensitive information (unique device id) that requires permission `READ_PHONE_STATE` by calling sensitive Android API named `getDeviceId()`. This information is encapsulated in Intent and sent to *Collector* app. Note that the *Collector* app does not have permission to access device id on its own, but due to inserted privacy leakage path, it can get the access. Then, the sensitive information get ex-filtrated to an external file. The sensitive information then ex-filtrate to an external file. This ex-filtration is without user consent as the user has not permitted *Colletor* app to access devide id information. The inserted leakage path cannot be detected by VirusTotal [4], AndroTotal [1], IccTA [12], FlowDroid [5] and Droidsafe [11]. The problem of collusion is dangerous in the sense that both the apps required a minimal set of permission and hence are treated as benign by the majority of available techniques. The challenge of collusion detection lies in representing sensitive data-flow by these apps and identifying the leakage path spread across multiple apps.

2. CONTRIBUTIONS

The contributions are as follows:

1. We propose a technique where verification method is used efficiently to check all the possible paths generated due to inter-app communication and to verify if the paths are admissible on the requirements of the safe state (no collusion).
2. In this work in progress, only intents have been explored as a means of inter-app communication mechanism. Based on this, our proposed collusion checking property can de-

ASIA CCS '17 April 02-06, 2017, Abu Dhabi, United Arab Emirates

ACM ISBN 978-1-4503-4944-4/17/04.

DOI: http://dx.doi.org/10.1145/3052973.3055163

Figure 1: Examples of the privacy leakage path spread across apps DeviceId, DeviceId_Service and Collector

tect presence/absence of collusion. To check for privilege escalation while colluding, Intents (that carry sensitive information) have been augmented with permissions.

3. We are proposing, a multi-app analysis tool for collusion detection.
4. We developed eight new apps that are diverse in the components used for communication, and type of Intent based communication channels (implicit, explicit, ordered). We also inserted sensitive leakage paths in these apps. These apps are contributed to the data set which can be used for the comparison of the techniques to detect privacy leakage through collusion.

3. PROPOSED APPROACH

In this section, we present the structure of our tool, which is designed to detect potentially colluding apps. Figure 2 shows the overview of our tool The description of the figure is as follows:

1. Android apps are implemented in Java and compiled into Dalvik bytecode. So in step 1, we extracts Java bytecodes from `.dex` files.
2. In step 2, we extract the main ICC classes like Intents, Intent Filters, and URIs along with Component Name, Bundle, Pending Intent and URI Builder classes.
3. In step 3, we extract methods corresponding to Sensitive Resource Access (methods that require dangerous permission).
4. Our tool stores all the collected information into a database for each app as shown in steps 4.1 and 4.2. Information is stored in the form of following tuple:
$$\langle intentID, intentAction, intentPerm \rangle$$
5. In step 5, we model the stored information by constructing App Promela Model for each app.
6. In step 6.1, these models are fed to SPIN Model Checker [3]. In step 6.2, we specify a collusion checking property in linear temporal logic (LTL) that says, the state of the model should always be **SAFE**: $[](state == SAFE)$
7. At last, in step 7, SPIN check all the paths exhaustively against the property. It will generate an error and provide a counterexample of the path that does not satisfy the LTL property. We will report that path in the colluding apps.

4. EVALUATION

In this section, we evaluate results from our experiments to judge the efficacy of our proposed tool. DroidBench [2] had three sample apps demonstrating inter-app communication through activity component. We developed eight new apps that exhibit collusion through inter-app communication. We open-sourced our experimental dataset of apps.

Experimental Results: To evaluate our approach, we launch our tool on DroidBench samples. We conduct our experiments in parts. Part I consists of two-apps scenario, in which we took two apps at a time and checked for the presence/absence of collusion due to their communication. While selecting two apps out of 8 apps, total possible tests cases are $\binom{8}{2} = 28$. But we reduce the number of test-cases by using prior information about the app. If there is no Intent communication between the apps, we can leave that test-case as they are not transferring any information. This reduces test-cases to 7 instead of 28. Table 1 summarizes the result of our analysis of the first scenario. Part II consists of 3-apps scenario, in which we analyze three apps simultaneously for collusion.

5. CONCLUSION

Private information leakage may pose a significant risk to the security of Android mobile users. Currently, most of the techniques target single-app analysis to detect privacy leakage path. However, the malicious app developers generate the leakage path across multiple apps. Hence it is challenging to detect such leakage paths.

This paper addresses the major challenges of multi-app analysis leading to information leakage. We presented our work in progress technique, a tool based on model checking for collusion detection. The proposal involves preprocessing on Android apps under analysis to extract relevant information. Extracted information can be further utilized to reduce the number of test-case for evaluation. It also helps in increasing the scalability of the tool. As a second broader step, Our method provides a formal representation of the extracted information. This step helps in a compact representation of relevant information that can be given to model-checking tool. In the end, our technique once developed shall apply model checking to verify if the collusion checking property is satisfied by the model or not. If not,

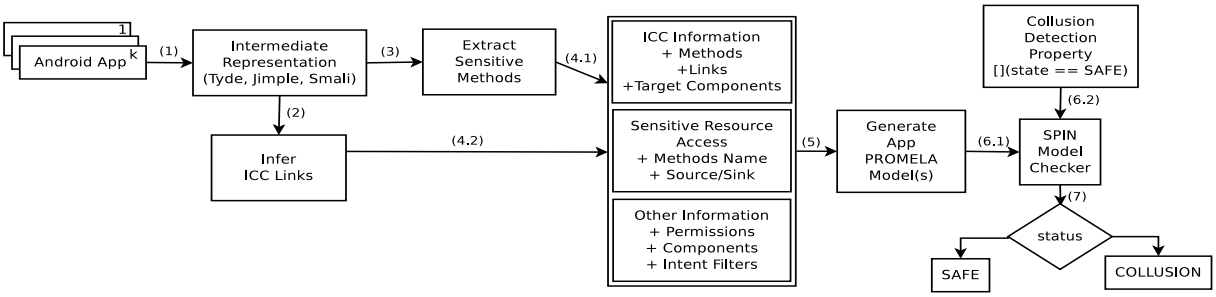

Figure 2: Workflow of our proposed tool

Apps	DeviceId_ Broadcast1	DeviceId_ Content Provider1	DeviceId_ Ordered Intent1	DeviceId_ Service1 Service1	Location1 Location1 Location1	Location_ Broadcast1 Broadcast1	Location_ Service1 Service1	Collector Collector Collector
DeviceId_ Broadcast1	-NA-	✉	✉	✉	✉	✉	✉	✓
DeviceId_ Content Provider1		-NA-	✉	✉	✉	✉	✉	✓
DeviceId_ Ordered Intent1			-NA-	✉	✉	✉	✉	✓
DeviceId_ Service1				-NA-	✉	✉	✉	✓
Location1					-NA-	✉	✉	✓
Location_ Broadcast1						-NA-	✉	✓
Location_ Service1							-NA-	✓
Collector								-NA-

✉: No Communication Exists, ✗: Absence of Collusion, ✓: Presence of Collusion, -NA- : Not Applicable

Table 1: Detection of collusion in two apps scenario from DroidBench

an alert is raised confirming the collusion among the apps under analysis.

6. ACKNOWLEDGMENTS

This work is partially supported by Security Analysis Framework for Android Platform (Grant 1000109932) by DeitY, Government of India. The work is also partially supported by DST-CNRS project IFC/DST-CNRS/2015-01/332 at MNIT Jaipur.

7. REFERENCES

[1] Androtoal. http://androtoal.org/. [Online; accessed 10-May-2015].

[2] DroidBench 2.0. https://github.com/secure-software-engineering/DroidBench. [Online; accessed 02-June-2015].

[3] SPIN Model Checker. http://www.spinroot.com. [Online; accessed 23-September-2015].

[4] Virustotal. http://virustotal.com/. [Online; accessed 10-May-2015].

[5] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. In *ACM SIGPLAN Notices*, volume 49, pages 259–269. ACM, 2014.

[6] S. Bhandari, W. B. Jaballah, V. Jain, V. Laxmi, A. Zemmari, M. S. Gaur, and M. Conti. Android app collusion threat and mitigation techniques. *arXiv preprint arXiv:1611.10076*, 2016.

[7] S. Bhandari, V. Laxmi, A. Zemmari, and M. S. Gaur. Intersection automata based model for android application collusion. In *2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA)*, pages 901–908. IEEE, 2016.

[8] S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, A. Sadeghi, and B. Shastry. Towards taming privilege-escalation attacks on android. In *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012*. The Internet Society, 2012.

[9] K. O. Elish, D. D. Yao, and G. R. Barbara. On the need of precise inter-app icc classificationfor detecting android malware collusions. In *Proceedings of the Security and Privacy Workshops*, pages 116–127, 2015.

[10] P. Faruki, S. Bhandari, V. Laxmi, M. Gaur, and M. Conti. Droidanalyst: Synergic app framework for static and dynamic app analysis. In *Recent Advances in Computational Intelligence in Defense and Security*, pages 519–552. Springer, 2016.

[11] M. I. Gordon, D. Kim, J. Perkins, L. Gilham, N. Nguyen, and M. Rinard. Information-flow analysis of android applications in droidsafe. In *Proc. of the Network and Distributed System Security Symposium (NDSS). The Internet Society*, 2015.

[12] L. Li, A. Bartel, T. F. D. A. Bissyande, J. Klein, Y. Le Traon, S. Arzt, S. Rasthofer, E. Bodden, D. Octeau, and P. McDaniel. Iccta: detecting inter-component privacy leaks in android apps. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE 2015)*, 2015.

[13] C. Marforio, H. Ritzdorf, A. Francillon, and S. Capkun. Analysis of the communication between colluding applications on modern smartphones. In *Proceedings of the 28th Annual Computer Security Applications Conference*, pages 51–60. ACM, 2012.