Software Target-Focused Flow Analysis

Anonymous

STATIC ANALYSIS

FlowDroid [2] is a sophisticated context-,flow-, field-, object-sensitive and lifecycle-aware static taint analysis tool for Android applications. It does not model the inter-component communication of Android applications and hence it cannot exam the implicit information flows. Epicc [8] statically resolves the destinations of each inter-component communication instance by reducing the problem into a traditional program analysis problem. IccTA [7] and DidFail [6] combine FlowDroid with Epicc and seek to recognize sensitive inter-component and inter-application information flows. DroidSafe [5] focus on providing more accurate modeling of Android system, which identifies some data flows missed by FlowDroid due to inaccuracy modeling of life-cycle and callback events.

Above approaches inherit the general limitations of static analysis i.e. they generate false alarms. These tools are oblivious to reflective calls, native code and multi-threading and generate false alarms. Also, in preliminary experiments running FlowDroid on real apps collected from the app markets, most of them terminate in unexpected ways: either due to timeout or out-of-memory exception.

DYNAMIC ANALYSIS

Dynamic analysis is embedded into the mobile operating systems and monitors the applications at runtime. TaintDroid [4] is a sophisticated dynamic taint-tracking tool. BayesDroid [9] is built upon TaintDroid to further classify information leakages through checking values between sources and sinks.

Compared to static analysis, dynamic analysis does not need to worry about the reflections inside the apps and only report the malicious behaviors during runtime, which generate much less false alarms and is more usable to the end users. However, low code coverage limits the overall detection rate of dynamic analysis.

Hybrid Analysis

To overcome the disadvantages and preserve the advantages of both static analysis and dynamic analysis, AppAudit [10] first attempts to model data flow inside apps by integrating static analysis and dynamic analysis. It quickly obtains rough over-estimated relationships between sources and sinks through static analysis, and then prunes the results through dynamic analysis. The outcomes show that it achieves better performance in both precision and usability, as compared to a pure static analysis approach or a dynamic analysis approach. Inspired by AppAudit, we attempt to come up with a better hybrid approach to extract program dependencies in the apps.

We notice that AppAudit still misses a lot data flows on DroidBench [1], which is a benchmark toolkit to test the performance of taint analysis tools. The main reason behind is AppAudit purely relies on concrete value to proceed the analysis. Unknown value, however, may be generated due to the insufficient run-time information. The existence of unknown value in the conditional program statements will lead to incomplete searching paths and inaccurate data flow results. The problem can be alleviated with the help of symbolic execution [3]. Symbolic execution leverages symbolic representation and first-order logic to represent an unknown variable. We will integrate the ideas from symbolic execution into the technique proposed by AppAudit to achieve higher accuracy.

Also, the call graph algorithm of AppAudit does not consider the potential sensitive behaviors inside Android GUI callbacks. We can create a more appropriate model of the mobile operating system to locate the stealthy behaviors.

Since the core engine of AppAudit is not open-source, we also need to implement our own virtual machine and program analysis module.

References

[1] Droidbench. `http://blogs.uni-paderborn.de/sse/tools/droidbench/`. Accessed: 2016-02-26.

[2] 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 *Proc. of PLDI*. ACM, 2014.

[3] C. Cadar and K. Sen. Symbolic execution for software testing: three decades later. *Communications of the ACM*, 56(2):82–90, 2013.

[4] W. Enck, P. Gilbert, S. Han, V. Tendulkar, 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. *ACM Transactions on Computer Systems (TOCS)*, 32(2):5, 2014.

[5] M. I. Gordon, D. Kim, J. H. Perkins, L. Gilham, N. Nguyen, and M. C. Rinard. Information flow analysis of android applications in droidsafe. In *NDSS*, 2015.

[6] W. Klieber, L. Flynn, A. Bhosale, L. Jia, and L. Bauer. Android taint flow analysis for app sets. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on the State of the Art in Java Program Analysis*, pages 1–6. ACM, 2014.

[7] L. Li, A. Bartel, T. F. Bissyandé, 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 *Proceedings of the 37th International Conference on Software Engineering-Volume 1*, pages 280–291. IEEE Press, 2015.

[8] D. Octeau, P. McDaniel, S. Jha, A. Bartel, E. Bodden, J. Klein, and Y. Le Traon. Effective inter-component communication mapping in android with epicc: An essential step towards holistic security analysis. *Effective Inter-Component Communication Mapping in Android with Epicc: An Essential Step Towards Holistic Security Analysis*, 2013.

[9] O. Tripp and J. Rubin. A bayesian approach to privacy enforcement in smartphones. In *Proc. of USENIX Security*, 2014.

[10] M. Xia, L. Gong, Y. Lyu, Z. Qi, and X. Liu. Effective real-time android application auditing. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 899–914. IEEE, 2015.