A General Approach to Modeling Java Framework Behaviors

Linghui Luo
linghui.luo@upb.de
Paderborn University
Germany

ABSTRACT

Interprocedural static analysis tools such as security analyses need good call graphs, which are challenging to scale for framework-based applications. So most tools model rather than analyzing frameworks. These models are manually crafted to capture framework semantics crucial for the particular analysis, and are inherently incomplete. We propose a general approach to modeling Java frameworks. It is not limited to any framework or analysis tool, therefore, highly reusable. While a generic approximation can be noisy, we show our carefully-constructed one does well. Experiments on Android with a client taint analysis show that our approach produces more complete call graphs than the original analysis. As a result, the client analysis works better: both precision (from 0.83 to 0.86) and recall (from 0.20 to 0.31) are improved.

CCS CONCEPTS

• **Theory of computation → Program analysis**.

KEYWORDS

call graph, static analysis, framework modeling, taint analysis, Java

ACM Reference Format:
Linghui Luo. 2021. A General Approach to Modeling Java Framework Behaviors. In *Proceedings of the 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '21), August 23–28, 2021, Athens, Greece.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3468264.3473489

1 INTRODUCTION

Call graphs are a key prerequisite of interprocedural static analyses. To be scalable, most static analysis tools choose to construct application-only call graphs and carefully model the behavior of frameworks [2, 3, 13, 16]. FlowDroid, for instance, models the behavior of the Android framework by crafting a dummy main method which simulates the lifecycle of Android components [3]. However, such carefully crafted models often produce incomplete call graphs, and are impractical to do for every framework [4, 14]. So we build a model that over-approximates to obviate framework details and yet retains enough detail for analysis. Particularly, we developed Averroes-GenCG—an improvement of Averroes [1] that generates a placeholder library for a given Android/Java framework. This generated library can be used as a replacement of the original framework by popular call graph construction algorithms [7, 15] and further client analyses. The framework behavior is modeled in the placeholder library code and will be reflected in the constructed call graphs.

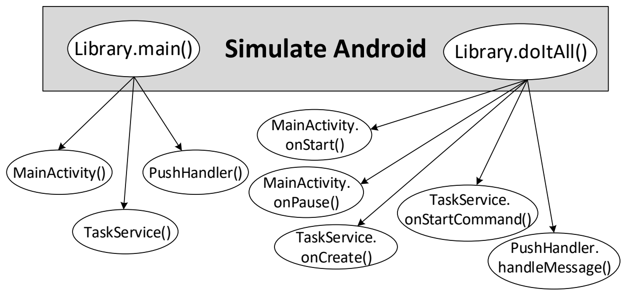

Figure 1: Simulating Android behavior.

2 BACKGROUND AND RELATED WORK

FlowDroid is a prominent static Android taint analysis tool; it performs context-, field-, and flow-sensitive data-flow analysis and delivers precise results. In comparison to other tools, FlowDroid achieves good result in evaluation on micro benchmarks such as DroidBench [10, 11]. We recently evaluated Android taint analysis tools on TaintBench [8, 9], which consists of 39 real-world malware apps and 203 documented malicious taint flows. Although FlowDroid produced the best result on TaintBench, it has especially low recall (0.2). Specifically, 35% (70/203) of the malicious taint flows in TaintBench could not be detected due to relevant methods being missing in the call graphs. Clearly, we need to construct better call graphs.

Our work was inspired by Averroes [1]. Averroes generates a placeholder library to overapproximate behavior of an original Java library. Averroes relies on the *separate compilation assumption*—the library can be compiled separately without the client application. Based on that, Averroes generates a `Library.doItAll()` method which implements behaviors such as object creation, invocation of library callbacks, etc.

3 APPROACH

The separate compilation assumption works for Android and other Java web frameworks. However, we could not directly use the placeholder.jar generated by Averroes for Android or web apps, since these apps do not have any main entry point. One could take the `Library.doItAll()` method as an entry point. However, it is less useful for detecting issues that requires flow-sensitivity, since `Library.doItAll()` contains no control flow at all and callbacks

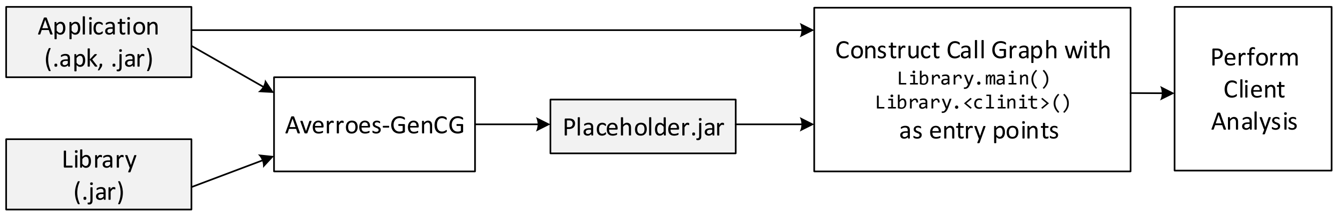

Figure 2: Overview of how to utilize our approach for a client analysis.

are unordered. Furthermore, Averroes uses a single `libraryPointsTo` field to represent all objects that the library may reference, which is too coarse for a field-sensitive taint analysis: once `libraryPointsTo` is tainted, it propagates everywhere, potentially resulting in many false positives. To produce call graphs suitable for flow- and field-sensitive client analyses, we adapt Averroes as follows:

- We introduce typed `libraryPointsTo` fields for every type `T` that the library could point to.
- We move object creation from `Library.doItAll()` to a separate `Library.main()` method to avoid unnecessary strong updates [5]. This simulates the framework as shown in Figure 1 for Android.
- We introduce control flow into `Library.doItAll()` to model other side effects of the library.

We refer to our adaption as Averroes-GenCG. Its generated placeholder.jar can be used by client analyses together with the application to construct call graphs with popular analysis frameworks [6, 12] as shown in Figure 2. Note that `Library.<clinit>()` is the static constructor of the `Library` class.

4 EVALUATION AND RESULTS

Table 1: Evaluation on TaintBench.

	FlowDroid	FlowDroid^{Gen}
TP	40	64
FP	8	10
FN (incomplete call graph)	70	19
FN (other reasons)	93	120
Precision p=TP/(TP+FP)	0.83	0.86
Recall r=TP/(TP+FN)	0.20	0.31
F-measure f=2pr/(p+r)	0.32	0.46

We evaluate our approach with FlowDroid^{Gen}, which is FlowDroid using our call graphs. We compare it to FlowDroid on TaintBench. Both tools were configured with sources and sinks that are used by the documented taint flows in each TaintBench app. The evaluation result is shown in Table 1. We explicitly distinguish false negatives (FN) caused by incomplete call graphs. A false negative can be caused by multiple factors. Here we mean incomplete call graphs is one factor. As we can see, 51 (row 4: 70 - 19) more malicious flows are captured in the call graphs of our approach in comparison to FlowDroid. For these 51 flows, our approach enabled FlowDroid's taint analysis to analyze all methods that are on their data-flow paths. Consequently, FlowDroid^{Gen} detected 24 (row 2: 64 - 40) more true positives (TP) with just 2 more false positives (FP) (row 3: 10 - 8). Thus, our call graphs improved both precision (from 0.83 to 0.86) and recall (from 0.20 to 0.31) of the client taint analysis.

5 CONCLUSION

Our approach is not limited to any framework or any specific analysis tool. Experiments on Android show our approach enables detection of more real-world issues without introducing much noise. As a next step, we will experiment our approach on Java web frameworks.

ACKNOWLEDGMENTS

This research was supported by the research training group Human Centered Systems Security (NERD.NRW) sponsored by the state of North Rhine-Westphalia in Germany.

REFERENCES

[1] Karim Ali and Ondrej Lhoták. 2013. Averroes: Whole-Program Analysis without the Whole Program. In *ECOOP 2013 - Object-Oriented Programming - 27th European Conference, Montpellier, France, July 1-5, 2013. Proceedings (Lecture Notes in Computer Science, Vol. 7920)*, Giuseppe Castagna (Ed.). Springer, 378–400. https://doi.org/10.1007/978-3-642-39038-8_16

[2] Anastasios Antoniadis, Nikos Filippakis, Paddy Krishnan, Raghavendra Ramesh, Nicholas Allen, and Yannis Smaragdakis. 2020. Static analysis of Java enterprise applications: frameworks and caches, the elephants in the room. In *Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2020, London, UK, June 15-20, 2020*, Alastair F. Donaldson and Emina Torlak (Eds.). ACM, 794–807. https://doi.org/10.1145/3385412.3386026

[3] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick D. McDaniel. 2014. FlowDroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps. In *PLDI '14, Edinburgh, United Kingdom - June 09 - 11, 2014*, Michael F. P. O'Boyle and Keshav Pingali (Eds.). ACM, 259–269. https://doi.org/10.1145/2594291.2594299

[4] Sam Blackshear, Alexandra Gendreau, and Bor-Yuh Evan Chang. 2015. Droidel: a general approach to Android framework modeling. In *Proceedings of the 4th ACM SIGPLAN International Workshop on State Of the Art in Program Analysis, SOAP@PLDI 2015, Portland, OR, USA, June 15 - 17, 2015*, Anders Møller and Mayur Naik (Eds.). ACM, 19–25. https://doi.org/10.1145/2771284.2771288

[5] Arnab De and Deepak D'Souza. 2012. Scalable Flow-Sensitive Pointer Analysis for Java with Strong Updates. In *ECOOP 2012 - Object-Oriented Programming - 26th European Conference, Beijing, China, June 11-16, 2012. Proceedings (Lecture Notes in Computer Science, Vol. 7313)*, James Noble (Ed.). Springer, 665–687. https://doi.org/10.1007/978-3-642-31057-7_29

[6] Raja Vallee-Rai et al. 1997. Soot. https://github.com/soot-oss/soot. Accessed: 2021-05-05.

[7] Ondrej Lhoták and Laurie J. Hendren. 2003. Scaling Java Points-to Analysis Using SPARK. In *Compiler Construction, 12th International Conference, CC 2003, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2003, Warsaw, Poland, April 7-11, 2003, Proceedings (Lecture Notes in Computer Science, Vol. 2622)*, Görel Hedin (Ed.). Springer, 153–169. https://doi.org/10.1007/3-540-36579-6_12

[8] Linghui Luo, Felix Pauck, Goran Piskachev, Manuel Benz, Ivan Pashchenko, Martin Mory, Eric Bodden, Ben Hermann, and Fabio Massacci. 2021. TaintBench. https://taintbench.github.io. Accessed: 2021-05-05.

[9] Linghui Luo, Felix Pauck, Goran Piskachev, Manuel Benz, Ivan Pashchenko, Martin Mory, Eric Bodden, Ben Hermann, and Fabio Massacci. 2021. TaintBench: Automatic Real-World Malware Benchmarking of Android Taint Analyses. *Empirical Software Engineering* (2021). to appear.

[10] Felix Pauck, Eric Bodden, and Heike Wehrheim. 2018. Do Android taint analysis tools keep their promises?. In *ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09, 2018*, Gary T. Leavens, Alessandro Garcia, and Corina S. Pasareanu (Eds.). ACM, 331–341. https://doi.org/10.1145/3236024.3236029

[11] Lina Qiu, Yingying Wang, and Julia Rubin. 2018. Analyzing the analyzers: FlowDroid/IccTA, AmanDroid, and DroidSafe. In *Proceedings of the 27th ISSTA*. ACM. https://doi.org/10.1145/3213846.3213873

[12] IBM Research. 2006. WALA. https://github.com/wala/WALA. Accessed: 2021-05-05.

[13] Manu Sridharan, Shay Artzi, Marco Pistoia, Salvatore Guarnieri, Omer Tripp, and Ryan Berg. 2011. F4F: taint analysis of framework-based web applications. In *Proceedings of the 26th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2011, part of SPLASH 2011, Portland, OR, USA, October 22 - 27, 2011*, Cristina Videira Lopes and Kathleen Fisher (Eds.). ACM, 1053–1068. https://doi.org/10.1145/2048066.2048145

[14] Li Sui, Jens Dietrich, Amjed Tahir, and George Fourtounis. 2020. On the recall of static call graph construction in practice. In *ICSE '20, Seoul, South Korea, 27 June - 19 July, 2020*, Gregg Rothermel and Doo-Hwan Bae (Eds.). ACM, 1049–1060. https://doi.org/10.1145/3377811.3380441

[15] Vijay Sundaresan, Laurie J. Hendren, Chrislain Razafimahefa, Raja Vallée-Rai, Patrick Lam, Etienne Gagnon, and Charles Godin. 2000. Practical virtual method call resolution for Java. In *Proceedings of the 2000 ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages & Applications (OOPSLA 2000), Minneapolis, Minnesota, USA, October 15-19, 2000*, Mary Beth Rosson and Doug Lea (Eds.). ACM, 264–280. https://doi.org/10.1145/353171.353189

[16] Fengguo Wei, Sankardas Roy, Xinming Ou, and Robby. 2014. Amandroid: A Precise and General Inter-component Data Flow Analysis Framework for Security Vetting of Android Apps. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, Gail-Joon Ahn, Moti Yung, and Ninghui Li (Eds.). ACM, 1329–1341. https://doi.org/10.1145/2660267.2660357