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.

CSC CONCEPTS

• Theory of computation → Program analysis.

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

ACM Reference Format:

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.

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
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 framework’s taint analysis to analyze all methods in 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


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


