Inferring Taint Specifications of Libraries

Chengpeng Wang

November 22, 2020

Dependencies on large libraries can degrade the effectiveness of static analysis significantly. Libraries frequently contain native code unable to be analyzed, use of challenging features including refection, and deep call hierarchies that reduce precision. As a specific static analysis, taint analysis depends on the integral value flow crossing the libraries, and a common-used workaround is to use taint specifications to summarize their behaviors. However, writing specifications for the entire library can be expensive and often error prone [1, 2]. In order to address these issues, approaches have been proposed to infer the taint specifications for library code automatically.

In this note, four types of approaches are discussed with a typical work. The lack of soundness and completeness of present works shows the value of lightweight shape analysis for the analysis of library code.

1 Problem

The problem is to infer taint specifications of library function contained in the program. A taint specification of library function is that the return value(an argument) is tainted if one of the arguments is tainted.

2 Related Work

2.1 Modelgen

Alex Aiken et al. propose Modelgen to apply a specified form of dynamic taint analysis to track the information flows between locations inside library functions [3]. By executing standard test cases of libraries, a partial collection of execution traces can be obtained, from which the data dependence is established, and then an unsound taint specification is inferred by lifting dynamic information flows to the flow between a method's arguments or between an argument and return value.

This work is based on dynamic taint analysis, and it suffers two drawbacks. Firstly, the specification is not sound and its quality depends on the coverage of code, which is affected by test cases.

Secondly, field-sensitivity is not assured in this work although it can be extended to generate field-sensitive taint specification. Specifically, lifting phase can be altered into the granularity of fields so that the taint flow in the form of $arg_i f1 \rightarrow ret f2$ can be stored in the specification.

2.2 StubDroid

Eric Bodden et al. present StubDroid to perform a taint analysis on selected public methods based on IFDS framework [4], and then generalize the resulting source-to-sink data-flow mappings and stores them in a summary file, which can be further utilized by FlowDroid [5].

This work is based on IFDS, a data flow analysis framework. Path insensitivity and its disadvantages of handling pointers make it not precise and efficient enough in the presence of multiple branches and intensive aliasing [6].

2.3 Atlas

Alex Aiken et al. propose Atlas to synthesize points-to specifications of a sequence of library function calls, namely path specifications, which support flow insensitive pointer analysis, so that information flow is connected [7]. Different from Modelgen and StubDroid, Atlas holds much weaker assumptions, namely available signatures and executable program. Potential path specifications are synthesized by Monte Carlo tree search based on compatible signatures and convinced by unit testing afterwards.

This work takes advantages of dynamic execution of library code as a black box as an evidence. There are two major drawbacks of this approach. Firstly, it abstract inner storage field as a ghost field making the result array-insensitive, such as Vector. Secondly, the soundness of specification can only be guaranteed if the client is flow insensitive, such as Andersen-style pointer analysis.

2.4 Other Works

There are a series of works which take advantages of auxiliary assumptions of client analysis to infer candidate specifications, such as the soundness assumption that true information flow must not be broken by the library code in each applications [8]. However, it can not assure the soundness nor completeness because it does not rely on any observation of library code, and it demands human interaction to check the specifications.

3 Conclusion

The restrictive form of available information on library code pose the challenge of sound specification inference. Dynamic execution-based and evidence-based inference only under-approximate the behavior of libraries and miss some unseen behaviors, while static analysis suffers the evil of false positives although soundness can be guaranteed in many instances. It might be possible to adapt shape analysis to the inference of a particular set of data structures for sound and precise specifications.

References

[1] Anastasios Antoniadis, Nikos Filippakis, Paddy Krishnan, Raghavendra Ramesh, Nicholas Allen, and Yannis Smaragdakis. Static analysis of java enterprise applications: frameworks and caches, the elephants in the room. In *PLDI*, pages 794–807, 2020.

[2] Pratik Fegade and Christian Wimmer. Scalable pointer analysis of data structures using semantic models. In *Proceedings of the 29th International Conference on Compiler Construction*, pages 39–50, 2020.

[3] Lazaro Clapp, Saswat Anand, and Alex Aiken. Modelgen: mining explicit information flow specifications from concrete executions. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, pages 129–140, 2015.

[4] Steven Arzt and Eric Bodden. Stubdroid: automatic inference of precise dataflow summaries for the android framework. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pages 725–735. IEEE, 2016.

[5] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. *Acm Sigplan Notices*, 49(6):259–269, 2014.

[6] Johannes Späth, Lisa Nguyen Quang Do, Karim Ali, and Eric Bodden. Boomerang: Demand-driven flow-and context-sensitive pointer analysis for java. In *30th European Conference on Object-Oriented Programming (ECOOP 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

[7] Osbert Bastani, Rahul Sharma, Alex Aiken, and Percy Liang. Active learning of points-to specifications. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 678–692, 2018.

[8] Osbert Bastani, Saswat Anand, and Alex Aiken. Specification inference using context-free language reachability. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 553–566, 2015.