Polyglot, Label-Defined Dynamic Taint Analysis in TruffleTaint

Jacob Kreindl
Johannes Kepler University Linz
Austria
jacob.kreindl@jku.at

Daniele Bonetta
Oracle Labs
Netherlands
daniele.bonetta@oracle.com

Lukas Stadler
Oracle Labs
Austria
lukas.stadler@oracle.com

David Leopoldseder
Oracle Labs
Austria
david.leopoldseder@oracle.com

Hanspeter Mössenböck
Johannes Kepler University Linz
Austria
hanspeter.moessenboeck@jku.at

ABSTRACT

Dynamic taint analysis assigns *taint labels* to sensitive data and tracks the propagation of such *tainted data* during program execution. This program analysis technique has been implemented in various analysis platforms targeting specific programming languages or program representations and has been applied to diverse fields such as software security and debugging. While some of these platforms support customization of their taint analysis, such customization is typically limited to certain analysis properties or to predefined options. This limitation can require analysis developers to modify the analysis platform in order to adapt other analysis properties or to implement new taint analysis applications.

We designed *label-defined dynamic taint analysis* as a new approach to specifying a dynamic taint analysis in terms of taint labels. This approach enables an analysis platform to allow analysis developers to adapt arbitrary analysis properties without modifying the platform itself. We implemented our approach in *TruffleTaint*, a *GraalVM*-based dynamic taint analysis platform targeting multiple programming languages. Our prototype supports implementing taint analyses in multiple programming languages and further provides tooling support for analysis development. In this tool demonstration we will present the capabilities of our prototype and demonstrate the implementation of label-defined dynamic taint analyses with common adaptations to various analysis properties.

CCS CONCEPTS

• **Software and its engineering** → **Application specific development environments**; *Dynamic analysis*; • **Security and privacy** → **Information flow control**.

KEYWORDS

Dynamic Taint Analysis, GraalVM, TruffleTaint

MPLR '22, September 14–15, 2022, Brussels, Belgium

ACM ISBN 978-1-4503-9696-7/22/09.
https://doi.org/10.1145/3546918.3560807

ACM Reference Format:
Jacob Kreindl, Daniele Bonetta, Lukas Stadler, David Leopoldseder, and Hanspeter Mössenböck. 2022. Polyglot, Label-Defined Dynamic Taint Analysis in TruffleTaint. In *Proceedings of the 19th International Conference on Managed Programming Languages and Runtimes (MPLR '22), September 14–15, 2022, Brussels, Belgium.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3546918.3560807

1 ADAPTABLE POLYGLOT DYNAMIC TAINT ANALYSIS USING TRUFFLETAINT

A *dynamic taint analysis* [3] is characterized by (1) its selection of *taint sources*, (2) the information contained in the *taint labels* it attaches to data that originates from these sources, (3) the *propagation semantics* defining when and which taint labels are transferred to data derived from tainted data, (4) its selection of *taint sinks*, and (5) which actions it sets when a tainted value flows into a taint sink.

We designed *label-defined dynamic taint analysis* [2], a language-agnostic approach to specify *all* these taint analysis properties in terms of taint labels. In our approach, taint labels are arbitrary objects that implement a special taint label interface. At run time, whenever a tainted value is accessed by an operation, the analysis platform calls methods of this interface on the value's taint label with additional information about the access as arguments. Analysis developers can implement these methods to affect the particular label's propagation semantics, to detect whether the operation accessing the value constitutes a taint sink, and, if so, to perform arbitrary actions. *Implicit labels* are special taint labels in our approach that can be implemented to be attached to the return values of all executed operations and to perform arbitrary actions before or after a particular operation is executed. While the use of implicit labels is optional, they enable analysis developers both to implement arbitrary taint sources and to support taint propagation over data flows that arise from control flow. Our approach further enables composition and instrumentation of taint analyses without requiring modifications to the analysis platform for these purposes.

Our prototype implementation in *TruffleTaint* supports label-defined dynamic taint analyses targeting multiple programming languages. While taint labels for TruffleTaint are generally implemented in Java, we also implemented a special one that uses GraalVM's polyglot capabilities to delegate calls of taint label methods to a taint label implemented in an arbitrary GraalVM-based, object-oriented programming language. We integrated TruffleTaint with GraalVM's language-agnostic debugger [1] to aid development of such polyglot taint analyses. We further implemented taint labels

which employ method delegation similar to the *polyglot label* to collect metadata useful for analysis development about an arbitrary other label-defined taint analysis.

2 TOOL DEMONSTRATION

In this tool demonstration we will show how to implement label-defined dynamic taint analyses and how to run them on TruffleTaint. Using a simple example program implemented in a combination of C and JavaScript, we will introduce the core concepts of dynamic taint analysis and demonstrate TruffleTaint's ability to propagate taint in both the program's two implementation languages and across the language boundary. In live coding we will then implement a label-defined dynamic taint analysis and discuss technical aspects of TruffleTaint. We will show how the methods of our taint label interface are invoked by the analysis platform and how they can be implemented to adapt the various properties that characterize a dynamic taint analysis. We will also implement common adaptations to these properties. Moreover, we will show how delegation of taint propagation decisions to a *wrapped analysis* enables both the implementation of configurable dynamic taint analysis applications and the instrumentation of other taint analyses to gather insights on their propagation. We will demonstrate as well that the same approach enables TruffleTaint to support implementing label-defined dynamic taint analyses in arbitrary object-oriented programming languages. During live coding we will also showcase TruffleTaint's integration with GraalVM's debugging infrastructure. We will further present additional tooling infrastructure we implemented that is specific to taint analysis development. The demonstration is expected to last about 45 minutes, not accounting for questions and discussion with the audience.

ACKNOWLEDGMENTS

This research project was partially funded by Oracle Labs. We thank all members of the Virtual Machine Research Group at Oracle Labs. Oracle, Java, GraalVM, and HotSpot are trademarks or registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners. We also thank all researchers at the Johannes Kepler University's Institute for System Software for their support of and feedback on our work.

REFERENCES

[1] M. L. Van de Vanter, C. Seaton, M. Haupt, C. Humer, and T. Würthinger. 2018. Fast, Flexible, Polyglot Instrumentation Support for Debuggers and other Tools. *Art Sci. Eng. Program.* 2, 3 (2018), 14. https://doi.org/10.22152/programming-journal.org/2018/2/14

[2] J. Kreindl, D. Bonetta, L. Stadler, D. Leopoldseder, and H. Mössenböck. 2022. Dynamic Taint Analysis with Label-Defined Semantics. In *MPLR '22: 19th ACM SIGPLAN International Conference on Managed Programming Languages and Runtimes, Brussels, Belgium, September 14-16, 2022*. ACM. https://doi.org/10.1145/3546918.3546927

[3] E. J. Schwartz, T. Avgerinos, and D. Brumley. 2010. All You Ever Wanted to Know about Dynamic Taint Analysis and Forward Symbolic Execution (but Might Have Been Afraid to Ask). In *31st IEEE Symposium on Security and Privacy, S&P 2010, 16-19 May 2010, Berleley/Oakland, California, USA*. IEEE Computer Society, 317–331. https://doi.org/10.1109/SP.2010.26