Jasa : A new Jasmin Safety Analyser based on MOPSA

Léo Juguet[1,2], Manuel Barbosa[3,4], Benjamin Grégoire[5], Vincent Laporte[6], and Raphaël Monat[7]

[1] ENS Paris-Saclay `leo.juguet@ens-paris-saclay.fr`
[2] MPI-SP, Bochum, Germany
[3] University of Porto (FCUP), Porto, Portugal `mbb@fc.up.pt`
[4] INESC TEC, Porto, Portugal
[5] Université Côte d'Azur, Inria, France `benjamin.gregoire@inria.fr`
[6] Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France `vincent.laporte@inria.fr`
[7] Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France `raphael.monat@inria.fr`

Abstract. We present a new security checker for Jasmin, a programming language for implementing high-assurance, high-speed cryptographic codes. The safety checker detects scalar initialization, array initialization to some extent, and out-of-bounds access without unrolling loops using widening. This new safety checker is mainly based on the MOPSA library. It is undeniably faster than its predecessor, thanks to support for function contracts and modular analysis. This new safety controller, created with MOPSA, also allows more properties to be checked than the previous safety controller, with the ability to analyse values.

Keywords: Abstract Interpretation · Array abstraction · Array content analysis · Program verification · Static analysis

1 Resume

1.1 Introduction

Context Writing a secure program is hard, as there are many considerations to take into account, and often small mistakes are involuntarily made by programmers, such as badly definining variables or accessing out-of-bound memory. These mistakes can lead to writing at unsafe locations, leading to an information leak. Hence, a tool is needed to detect mistakes.

In particular for Jasmin [1] a programming language that aims to be secure for cryptographic code. Its compiler is mostly written in Coq and provides a proof that the code will be correctly translated to assembly, under certain assumptions. These assumptions are checked by a safety checker, that use abstract interpretation [2] but the current tool is too slow, and not as precise as we wanted. Moreover the safety checker is hard to maintain today, and doesn't allow modular analysis.

The goal of this work was to create a new safety checker more efficient, more maintanable, and more precise safety analyzer that would be able to check that safety conditions of any Jasmin code are verified.

Contribution We wrote a new safety checker for Jasmin [8][1] using the MOPSA [4] library . MOPSA[4] is a modular open platform for static analysis that uses abstract interpretation and is designed to be modular, so that a frontend for different languages can easily be added. By relying on the third-party MOPSA library, which maintains the abstract interpretation backend, the resulting code base is easier to maintain from Jasmin's point of view.

This new safety checker can check whether scalars and arrays are correctly initialized. The analysis of array initialisation is based on an algorithm mainly inspired by a paper by Cousot [3], which cannot currently be implemented in MOPSA. This algorithm takes into account two main criteria: a limited number of variables needed to represent an array and it is not necessary to unfold loops when analysing arrays. In the end, all arrays are represented by three segments and the abstraction of cell values is not limited to an initialisation domain.

In addition, this safety checker can detect out-of-bounds errors in arrays and is modular, allowing function-independent analysis. Thanks to contract support for checking properties, although at present only contracts for array initialisation have been implemented. It also works faster than the current checker.

Limits However, for the moment, the tool cannot be used in a complete Jasmin code base due to the current lack of support for special instructions and memory. But support for special instructions can easily be added with the work already done on function calls. Memory analysis is more difficult because memory aliasing has to be taken into account.

Acknowledgments. This work was produced during 5-month internship at MPI-SP, in the Jasmin compiler team.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Almeida, J.B., Barbosa, M., Barthe, G., Blot, A., Grégoire, B., Laporte, V., Oliveira, T., Pacheco, H., Schmidt, B., Strub, P.Y.: Jasmin: High-assurance and high-speed cryptography. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. p. 1807–1823. CCS '17, Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3133956.3134078, https://doi.org/10.1145/3133956.3134078

[8] The source code can be found here : https://github.com/LeoJuguet/jasmin/tree/cryptoline-mopsa/compiler/jasa

2. Cousot, P., Cousot, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages. p. 238–252. POPL '77, Association for Computing Machinery, New York, NY, USA (1977). https://doi.org/10.1145/512950.512973, https://doi.org/10.1145/512950.512973
3. Cousot, P., Cousot, R., Logozzo, F.: A parametric segmentation functor for fully automatic and scalable array content analysis. SIGPLAN Not. **46**(1), 105–118 (jan 2011). https://doi.org/10.1145/1925844.1926399, https://doi.org/10.1145/1925844.1926399
4. Monat, R.: Static type and value analysis by abstract interpretation of Python programs with native C libraries. Ph.D. thesis (2021), http://www.theses.fr/2021SORUS263, thèse de doctorat dirigée par Miné, Antoine Informatique Sorbonne université 2021