Supporting Secure Coding with RefactorErl[3]

Brigitta Baranyai, Melinda Tóth, István Bozó

ELTE, Eötvös Loránd University
Faculty of Informatics
Department of Programming Languages and Compilers
ljcy93@inf.elte.hu, toth_m@inf.elte.hu, bozo_i@inf.elte.hu

In the era of the Internet there is a continuous demand for distributed systems which should serve thousands of requests on a daily basis. But with this growing demand companies have to face a growing number of cyber threats as well which can not only harm their customers in the form of data theft or data loss but their own reputation as well. In order to improve the security of the systems [**1**], there are several standards (CERT, OWASP's Application Security Verification Standard) and static analyser tools (CodeChecker, SpotBugs, SonarQube, Fortify) to achieve this goal. Unfortunately, these tools only cover popular programming languages like C++, Java, Python or TypeScript.

Certainly a lot has changed since Erlang [**2**] was initially introduced in the 80s. For any language to remain relevant it is a necessity to adjust to these new requirements imposed by the industry. It is important to try to close this gap that currently exists in the domain of static security analysers in order to keep Erlang as relevant as it is today and to possibly increase its popularity in the programming industry.

The change of the original concept of running Erlang on protected hosts, and the human factor (e.g. inexperienced programmers, the lack of knowledge about a specific application) of the development process makes it necessary to consider secure coding during the development of Erlang applications as well [**3**, **4**, **5**, **6**].

The main contributions of this paper is providing methodology and tool for the Erlang community to help identifying security vulnerabilities [**9**, **10**] like OS injection, cryptography or atom exhaustion related attacks in an early phase of the development process, therefore improving the security level of their Erlang applications. We examined and categorized the potential vulnerabilities specific to Erlang and based on these categories we defined checkers to identify the source code fragments that violates the security rules. The checkers were defined using a set of static source code analyses (e.g. data-flow, call-graph information) provided by RefactorErl [**7**, **8**].

References

[1] E. Jaeger and O. Levillain *Mind Your Language(s): A Discussion about Languages and Security.* In 2014 IEEE Security and Privacy Workshops, pages 140–151, 2014.

[2] Fred Herbert: Learn You Some Erlang for Great Good!: A Beginner's Guide, No Starch Press, San Francisco, CA, USA, 2013, ISBN 1593274351, ISBN 9781593274351

[3] Alexandre Jorge Barbosa Rodrigues and Viktória Fördős. *Towards secure 2 systems.* In Proceedings of the 17th ACM SIGPLAN International Workshop on Erlang, Erlang 2018, page 67–70, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450358248.

[3] The research has been supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.3-VEKOP-16- 2017-00002). The research is part of the "Application Domain Specific Highly Reliable IT Solutions" project that has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the Thematic Excellence Programme no. 2020-4.1.1.-TKP2020 (National Challenges Subprogramme) funding scheme.

[4] Fördős, Viktória: Secure Design and Verification of Erlang Systems, Proceedings of the 19th ACM SIGPLAN International Workshop on Erlang, pages 31–40, Association for Computing Machinery, New York, NY, USA, https://doi.org/10.1145/3406085.3409011, ISBN 9781450380492, 2020

[5] Security Working Group of the Erlang Ecosystem Foundation. *Secure coding and deployment hardening guidelines.* `https://erlef.github.io/security-wg/, 2020`.

[6] Kenji Rikitake. *Shared Nothing Secure Programming in Erlang/OTP*. In IEICE Tech. Rep., volume 114 of IA2014-11, ICSS2014-11, pages 55–60, Hyogo, June 2014, Takikawa Memorial Hall, Kobe University (ICSS, IA).

[7] István Bozó, Dániel Horpácsi, Zoltán Horváth, Róbert Kitlei, Judit Kőszegi, Máté Tejfel, and Melinda Tóth. *Refactorerl - Source Code Analysis and Refactoring in Erlang.* In Proceedings of the 12th Symposium on Programming Languages and Software Tools, ISBN 978-9949-23-178-2, pages 138–148, Tallin, Estonia, October 2011.

[8] Melinda Tóth and István Bozó. *Static Analysis of Complex Software Systems Implemented in Erlang.* Central European Functional Programming Summer School - Fourth Summer School, CEFP 2011, Revisited Selected Lectures, Lecture Notes in Computer Science (LNCS), Vol. 7241, pp. 451-514, Springer-Verlag, ISSN: 0302-9743.

[9] Brigitta Baranyai: Funkcionális nyelvek és a statikus kódelemzéssel támogatott biztonságos szoftverfejlesztés, Paper at the Student Association Conference, Faculty of Informatics, Eötvös Loránd University, May 2020, Received 2nd prize.

[10] Brigitta Baranyai, Melinda Tóth, István Bozó: Supporting Secure Coding with RefactorErl, Talk at the 19th ACM SIGPLAN International Workshop on Erlang, Virtual Event, 23 August, 2020

Diagnosing vulnerabilities with static analysis[2]

Ábel Kocsis, Zoltán Gera, Melinda Tóth

Department of Programming Languages and Compilers
Faculty of Informatics
ELTE, Eötvös Loránd University

abelkocsis@caesar.elte.hu, gerazo@inf.elte.hu, toth_m@inf.elte.hu

Poorly-designed programs often cause errors and failures, which may lead to an irreversible disaster, such as the Ariane 501 satellite launch [**1**]. A software which is not designed thoroughly is usually more vulnerable, so attackers may manage to steal personal data from customers much more easily. To avoid these mistakes, there are numerous static analyser tools which are able to check the programs without executing them. This is crucial, since the earlier an error is discovered, the easier and cheaper it is to fix it [**2**]. However, at first sight, it is really hard for a programmer or a company to tell which tool to use.

In this research, we aim to answer this question by comparing the most popular open-source static analyser tools for C and C++ language. Different comparative studies have been conducted in the field of static analysers before [**3**, **4**]. However, these papers do not consider other factors than the results of a few analysers.

A previous study [**5**] has examined the reasons for using and underusing static analysis tools. These reasons are tool output, collaboration, customisability and result understandability. Since our study concentrates on open-source tools, two other aspects of comparison have been added: method of analysis and collaboration. To get a full picture of each tool, testing for the most common vulnerabilities have been added to the factors. These aspects have made it possible to examine the tools in-depth and to spot the key features of each tools, analysis or a wide range of available checkers, respectively.

Eleven of the most popular static analysis tools [**6**], have been compared. These tools are the following: Clang Static Analyzer, Clang-Tidy, Cppcheck, Facebook Infer, Flawfinder, Ikos, RATS, Smatch, Splint, SVF and Yasca. The tools differ from each other in a number of important characteristics. These programs are quite diverse; there are vulnerabilities which can be diagnosed with them, and there are many which may be impossible to spot. In the end of the research, a recommendation has been stated which tool may be suitable for general use and which should not be used.

LLVM Clang Static Analyzer and Clang-Tidy have been found to be one of the best analyzers in this comparison. After that, four new checkers have been developed for Clang-Tidy to diagnose hardly-recognisable vulnerabilities connected to multithreaded C and C++ programs; and another checker to improve Clang Static Analyzer.

References

[1] Mark Dowson. 1997. The Ariane 5 software failure. *SIGSOFT Softw. Eng. Notes 22, 2 (March 1997), 84.* DOI:https://doi.org/10.1145/251880.251992

[2] N. Nagappan and T. Ball, "Static analysis tools as early indicators of pre-release defect density," *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005., Saint Louis, MO, USA, 2005,* pp. 580-586, doi: 10.1109/ICSE.2005.1553604.

[3] Rahma Mahmood and Qusay H. Mahmoud. Evaluation of static analysis tools for finding vulnerabilities in java and C/C++ source code. CoRR, abs/1805.09040, 2018.

[2] The research has been supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.3-VEKOP-16- 2017-00002), and by the ÚNKP-19-1 New National Excellence Program of Ministry for Innovation and Technology. The research is part of the "Application Domain Specific Highly Reliable IT Solutions" project that has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the Thematic Excellence Programme no. 2020-4.1.1.-TKP2020 (National Challenges Subprogramme) funding scheme.

[4] Pär Emanuelsson and Ulf Nilsson. A comparative study of industrial static analysis tools. Electronic Notes in Theoretical Computer Science, 217:5 – 21, 2008. Proceedings of the 3rd International Workshop on Systems Software Verification (SSV 2008).

[5] B. Johnson, Y. Song, E. Murphy-Hill and R. Bowdidge, "Why don't software developers use static analysis tools to find bugs?," *2013 35th International Conference on Software Engineering (ICSE)*, San Francisco, CA, 2013, pp. 672-681, doi: 10.1109/ICSE.2013.6606613.

[6] Ábel Kocsis: Programok sebezhetőségének felismerése statikus kódelemzéssel, Paper at the Student Association Conference, Faculty of Informatics, Eötvös Loránd University, May 2020, Received 1st prize.