Research Internship 2021-2022 Topic

Security counter-measures in a certified optimizing compiler

Supervisors: Sylvain Boulmé and Marie-Laure Potet at Verimag*

mailto:Sylvain.Boulme@univ-grenoble-alpes.fr

CompCert[1] is a compiler for the C programming language for the assembly languages of several processor architectures. In contrast to compilers such as Visual C++, GCC, or LLVM, its compilation phases are proved mathematically correct, and thus the compiled program always matches the source program: the formal correctness of CompCert states that if the compiler succeeds to produce an executable, then the *observable behaviors* of the executable are also *observable* on the source program [1, 2]. Other compilers may contain bugs that in some cases result in incorrect code being generated. The possibility of compilation bug cannot be tolerated in certain applications with high safety requirements, and then costly solutions such as disabling all optimizations are used to get assembly code that is close to the source. In contrast, CompCert, despite not optimizing as well as gcc -O3 or clang -O3, allows using optimizations safely [3, 4].

Security of embedded systems like smart-cards, secured dongles and IoTs relies on the robustness of devices against physical fault attacks (such as laser or electromagnetic attacks). Current certification schemes (e.g., Common Criteria) require protection mechanisms against multi-fault injection attacks. Typically, these protections often consist in Counter-Measures (CM): monitors which perform redundant computations in order to detect/prevent some attacks.

Some of these CMs are "manually" written in the source code by developers. For example, the process of counting the number of trials for typing some pin-code is duplicated in order to make successful hardware attacks on this number more complex. However, because such CM perform redundant computations on execution without attacks, optimizing compilers may remove them.

A solution of Vu-et-al [5, 6, 7] has been experimented within LLVM compiler. It consists in introducing observations of the program state that are intrinsic to the correct execution of security protections, along with means to specify and preserve observations across the compilation flow. Such observations complement the input/output semantics-preservation contract of compilers. In practice, they are given as annotations of the source code.

From these works, the internship will study how to use the formal notion of *observable events* of CompCert, in order to ensure that CMs are not removed by the compiler (as a consequence of its formal correctness). Here is a list of related questions:

- For case studies from the state-of-the-art (e.g. [8]) introducing particular CM with respect to some given security goals, which guarantees are provided by the CM (e.g.

*http://www-verimag.imag.fr/

[1]https://compcert.org/

under which hypotheses setting to zero a given memory buffer forbids the leak of a given secret information)? Which observations/annotations are necessary to ensure their preservation by the compilation chain?

- How can we use the annotations to evaluate the necessity and the efficiency of the CM with respect to the security goals (for example, by reusing a tool like Lazart [9, 10])?
- Does the formal semantics of CompCert helps to formalize (at least partially) the guarantees provided by the CM wrt security goals?
- How this approach, in which the programmer specifies how to ensure the security goals, compares with other approches, such as [11, 12, 13, 14, 15, 16], where CompCert automatically ensures some security goals? Is the annotation approach more lighweight for the compiler design, but more heavyweight for the programmer? It is more general?

This wide topic is also the subject of a future Phd thesis proposed as an extension to the internship.

References

[1] X. Leroy, "Formal verification of a realistic compiler," *Communications of the ACM*, vol. 52, no. 7, 2009. HAL: inria-00415861.

[2] X. Leroy, "A formally verified compiler back-end," *Journal of Automated Reasoning*, vol. 43, no. 4, pp. 363–446, 2009. [Online]. Available: http://xavierleroy.org/publi/compcert-backend.pdf.

[3] R. Bedin França, S. Blazy, D. Favre-Felix, X. Leroy, M. Pantel, and J. Souyris, "Formally verified optimizing compilation in ACG-based flight control software," Anglais, in *Embedded Real Time Software and Systems (ERTS2)*, AAAF, SEE, Feb. 2012. HAL: hal-00653367.

[4] D. Kästner *et al.*, "CompCert: Practical Experience on Integrating and Qualifying a Formally Verified Optimizing Compiler," in *ERTS2 2018 - 9th European Congress Embedded Real-Time Software and Systems*, 3AF, SEE, SIE, Toulouse, France, Jan. 2018, pp. 1–9. [Online]. Available: https://hal.inria.fr/hal-01643290.

[5] S. T. Vu, K. Heydemann, A. de Grandmaison, and A. Cohen, "Secure delivery of program properties through optimizing compilation," in *CC '20: 29th International Conference on Compiler Construction, San Diego, CA, USA, February 22-23, 2020*, L. Pouchet and A. Jimborean, Eds., ACM, 2020, pp. 14–26. DOI: 10.1145/3377555.3377897. [Online]. Available: https://doi.org/10.1145/3377555.3377897.

[6] S. T. Vu, A. Cohen, K. Heydemann, A. de Grandmaison, and C. Guillon, "Secure optimization through opaque observations," *CoRR*, vol. abs/2101.06039, 2021. arXiv: 2101.06039. [Online]. Available: https://arxiv.org/abs/2101.06039.

[7] S. T. Vu, A. Cohen, A. D. Grandmaison, C. Guillon, and K. Heydemann, "Reconciling optimization with secure compilation," 2021. [Online]. Available: https://research.google/pubs/pub50686/.

[8] L. Dureuil, G. Petiot, M. Potet, T. Le, A. Crohen, and P. de Choudens, "FISSC: A fault injection and simulation secure collection," in *Computer Safety, Reliability, and Security - 35th International Conference, SAFECOMP 2016, Trondheim, Norway, September 21-23, 2016, Proceedings*, A. Skavhaug, J. Guiochet, and F. Bitsch, Eds., ser. Lecture Notes in Computer Science, vol. 9922, Springer, 2016, pp. 3–11. DOI: 10.1007/978-3-319-45477-1_1. [Online]. Available: https://doi.org/10.1007/978-3-319-45477-1_1.

[9] M. Potet, L. Mounier, M. Puys, and L. Dureuil, "Lazart: A symbolic approach for evaluation the robustness of secured codes against control flow injections," in *Seventh IEEE International Conference on Software Testing, Verification and Validation, ICST 2014, March 31 2014-April 4, 2014, Cleveland, Ohio, USA*, IEEE Computer Society, 2014, pp. 213–222. DOI: 10.1109/ICST.2014.34. [Online]. Available: https://doi.org/10.1109/ICST.2014.34.

[10] E. Boespflug, C. Ene, L. Mounier, and M. Potet, "Countermeasures optimization in multiple fault-injection context," in *17th Workshop on Fault Detection and Tolerance in Cryptography, FDTC 2020, Milan, Italy, September 13, 2020*, IEEE, 2020, pp. 26–34. DOI: 10.1109/FDTC51366.2020.00011. [Online]. Available: https://doi.org/10.1109/FDTC51366.2020.00011.

[11] F. Besson, T. P. Jensen, and J. Lepiller, "Modular software fault isolation as abstract interpretation," in *Static Analysis - 25th International Symposium, SAS 2018, Freiburg, Germany, August 29-31, 2018, Proceedings*, A. Podelski, Ed., ser. Lecture Notes in Computer Science, vol. 11002, Springer, 2018, pp. 166–186. DOI: 10.1007/978-3-319-99725-4_12. [Online]. Available: https://doi.org/10.1007/978-3-319-99725-4_12.

[12] F. Besson, A. Dang, and T. P. Jensen, "Securing compilation against memory probing," in *Proceedings of the 13th Workshop on Programming Languages and Analysis for Security, PLAS@CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, M. S. Alvim and S. Delaune, Eds., ACM, 2018, pp. 29–40. DOI: 10.1145/3264820.3264822. [Online]. Available: https://doi.org/10.1145/3264820.3264822.

[13] F. Besson, S. Blazy, A. Dang, T. P. Jensen, and P. Wilke, "Compiling sandboxes: Formally verified software fault isolation," in *Programming Languages and Systems - 28th European Symposium on Programming, ESOP 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings*, L. Caires, Ed., ser. Lecture Notes in Computer Science, vol. 11423, Springer, 2019, pp. 499–524. DOI: 10.1007/978-3-030-17184-1_18. [Online]. Available: https://doi.org/10.1007/978-3-030-17184-1_18.

[14] F. Besson, A. Dang, and T. P. Jensen, "Information-flow preservation in compiler optimisations," in *32nd IEEE Computer Security Foundations Symposium, CSF 2019, Hoboken, NJ, USA, June 25-28, 2019*, IEEE, 2019, pp. 230–242. DOI: 10.1109/CSF.2019.00023. [Online]. Available: https://doi.org/10.1109/CSF.2019.00023.

[15] G. Barthe *et al.*, "Formal verification of a constant-time preserving C compiler," *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 926, 2019. [Online]. Available: https://eprint.iacr.org/2019/926.

[16] S. Blazy, D. Pichardie, and A. Trieu, "Verifying constant-time implementations by abstract interpretation," *J. Comput. Secur.*, vol. 27, no. 1, pp. 137–163, 2019. DOI: 10.3233/JCS-181136. [Online]. Available: https://doi.org/10.3233/JCS-181136.