Annotations de sécurité pour compilateur optimisant formellement vérifié.

CompCert compile des programmes C en langage assembleur de plusieurs processeurs. Contrairement aux autres compilateur comme Visual C++, GCC ou LLVM, ses phases de compilation sont prouvées correctes avec un vérificateur de théorèmes (en l'occurrence Coq). Ces autres compilateurs peuvent contenir des bugs, dits de « miscompilation », conduisant à produire du code incorrect. De tels bugs peuvent être dramatiques dans les applications critiques et conduisent parfois à les concepteurs de telles applications à compiler sans optimisation pour vérifier "manuellement" que le code assembleur correspond au code source. CompCert, bien que ne réalisant pas toutes les optimisations de ses concurrents, optimise les programmes assembleurs générés avec un niveau de sûreté inégalé [1, 2, 3, 4, 5].

Mais, face à des attaques de sécurité, les optimisations réalisées par le compilateur, même quand elles sont fonctionnellement correctes, peuvent être indésirables. Typiquement, elles peuvent invalider (voire éliminer) des contre-mesures logicielles, comme la mise à zéro de données sensibles ou les calculs redondants détectant les incohérences induites par attaque matérielle. Or, ces contre-mesures sont plus faciles à mettre en place lors de la conception du logiciel (i.e. au niveau source) : idéalement elles doivent donc ête transportées du source au binaire. De plus, il serait intéressant de mettre en lien (par exemple par un système d'annotations) les contre-mesures et leur objectif de sécurité.

Des premiers travaux dans ce sens ont été développés dans la thèse de Son Tuan Vu[6, 7] autour du compilateur LLVM. Le principe est de poser des observateurs imposant au compilateur de préserver des valeurs en un point du programme et de rendre opaque certaines valeurs de calcul afin d'empêcher certaines optimisations.

Le point de départ de cette thèse sera de reprendre ces travaux et de voir comment les intégrer dans CompCert, sachant que CompCert propose déjà des systèmes d'annotations qui garantissent la préservation de certaines observations par le code généré (avec une preuve formelle de cette préservation dans les passes successives du compilateur). En particulier,

- Il faudra mener des études de cas (inspirée de l'état de l'art, comme [8]) sur certains objectifs de sécurité et certaines contre-mesures : quelles garanties les contre-mesures permettent d'assurer face aux objectifs de sécurité (e.g. sous quelles hypothèses la mise à zéro d'une mémoire garantit l'absence de fuite d'informations sensibles) ? quelles observations faut-il faire sur ces contre-mesures afin d'assurer leur préservation par la chaîne de compilation ?
- Quelles contre-mesures/objectifs de sécurités sont difficilement exprimables dans le système d'annotations de Vu ? Comment étendre celui-ci ?
- Comment utiliser le mécanisme d'annotations pour évaluer l'efficacité et la pertinence des contre-mesures vis-à-vis des objectifs de sécurité (en lien par exemple, avec un outil comme Lazart [9, 10]) ?
- La sémantique formelle offerte par CompCert permet-elle de formaliser (au moins partiellement) les garanties offertes par les contre-mesures vis-à-vis des objectifs de sécurité ?
- Comment cette approche, où c'est le programmeur qui indique comment garantir l'objectif de sécurité, se compare avec d'autres approches, comme [11, 12, 13, 14, 15, 16], où c'est CompCert qui prend en charge les objectifs de sécurité ? A priori, on s'attend à ce que l'approche par annotation soit plus facilement généralisable à divers objectifs de sécurité, et que la mise en œuvre soit moins lourde au niveau du développement de CompCert (quitte à demander plus de travail à l'utilisateur de CompCert). À quel point ce point-de-vue à priori se vérifie-t-il en pratique ?

Références

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

[2] ——, “A formally verified compiler back-end”, *Journal of Automated Reasoning*, t. 43, n° 4, p. 363-446, 2009. adresse : http://xavierleroy.org/publi/compcert-backend.pdf.

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

[4] D. Kästner, J. Barrho, U. Wünsche, M. Schlickling, B. Schommer, M. Schmidt, C. Ferdinand, X. Leroy et S. Blazy, “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, p. 1-9. adresse : https://hal.inria.fr/hal-01643290.

[5] C. Six, S. Boulmé et D. Monniaux, “Certified and Efficient Instruction Scheduling : Application to Interlocked VLIW Processors”, *Proc. ACM Program. Lang.*, t. 4, n° OOPSLA, nov. 2020. doi : 10.1145/3428197. adresse : https://hal.archives-ouvertes.fr/hal-02185883.

[6] S. T. Vu, K. Heydemann, A. de Grandmaison et 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.-N. Pouchet et A. Jimborean, éd., ACM, 2020, p. 14-26. doi : 10.1145/3377555.3377897. adresse : https://doi.org/10.1145/3377555.3377897.

[7] S. T. Vu, A. Cohen, K. Heydemann, A. de Grandmaison et C. Guillon, “Secure Optimization Through Opaque Observations”, *CoRR*, t. abs/2101.06039, 2021. arXiv : 2101.06039. adresse : https://arxiv.org/abs/2101.06039.

[8] L. Dureuil, G. Petiot, M.-L. Potet, T.-H. Le, A. Crohen et 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 et F. Bitsch, éd., sér. Lecture Notes in Computer Science, t. 9922, Springer, 2016, p. 3-11. doi : 10.1007/978-3-319-45477-1_1. adresse : https://doi.org/10.1007/978-3-319-45477-1_1.

[9] M.-L. Potet, L. Mounier, M. Puys et 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, p. 213-222. doi : 10.1109/ICST.2014.34. adresse : https://doi.org/10.1109/ICST.2014.34.

[10] E. Boespflug, C. Ene, L. Mounier et M.-L. 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, p. 26-34. doi : 10.1109/FDTC51366.2020.00011. adresse : https://doi.org/10.1109/FDTC51366.2020.00011.

[11] F. Besson, T. P. Jensen et 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, éd., sér. Lecture Notes in Computer Science, t. 11002, Springer, 2018, p. 166-186. doi : 10.1007/978-3-319-99725-4_12. adresse : https://doi.org/10.1007/978-3-319-99725-4_12.

[12] F. Besson, A. Dang et 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 et S. Delaune, éd., ACM, 2018, p. 29-40. doi : 10.1145/3264820.3264822. adresse : https://doi.org/10.1145/3264820.3264822.

[13] F. Besson, S. Blazy, A. Dang, T. P. Jensen et 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, éd., sér. Lecture Notes in Computer Science, t. 11423, Springer, 2019, p. 499-524. doi : 10.1007/978-3-030-17184-1_18. adresse : https://doi.org/10.1007/978-3-030-17184-1_18.

[14] F. Besson, A. Dang et 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, p. 230-242. doi : 10.1109/CSF.2019.00023. adresse : https://doi.org/10.1109/CSF.2019.00023.

[15] G. Barthe, S. Blazy, B. Grégoire, R. Hutin, V. Laporte, D. Pichardie et A. Trieu, "Formal Verification of a Constant-Time Preserving C Compiler", *IACR Cryptol. ePrint Arch.*, t. 2019, p. 926, 2019. adresse : https://eprint.iacr.org/2019/926.

[16] S. Blazy, D. Pichardie et A. Trieu, "Verifying constant-time implementations by abstract interpretation", *J. Comput. Secur.*, t. 27, n° 1, p. 137-163, 2019. doi : 10.3233/JCS-181136. adresse : https://doi.org/10.3233/JCS-181136.