Vérification formelle d'une carte à puce pour une certification Critères Communs*

Adel Djoudi[1], Martin Hána[2], and Nikolai Kosmatov[3]

[1]Thales Digital Identity & Security, Meudon, France
[2]Thales Digital Identity & Security, Prague, Czech Republic
[3]Thales Research & Technology, Palaiseau, France,
firstname.lastname@thalesgroup.com

1 Contexte et motivation

La sûreté et la sécurité des logiciels critiques sont devenues aujourd'hui des préoccupations majeures. La vérification formelle de programme peut fournir une preuve rigoureuse de l'absence d'erreurs et de vulnérabilités de sécurité. Elle peut assurer des garanties de correction très fortes qui sont particulièrement importantes pour les produits critiques, tels que les cartes à puce. Cependant, la vérification formelle des logiciels industriels reste un défi.

La sécurité d'une carte à puce repose sur un ensemble de propriétés d'isolation qui doivent être assurées par la machine virtuelle de la carte (*JavaCard virtual machine*, ou JCVM). Selon la politique de sécurité [2], une application ne doit pas lire ou écrire les données ou exécuter le code d'une autre application sans son autorisation. Les règles d'accès correspondantes sont en général assurées par un mécanisme de contrôle d'accès dédié, appelé pare-feu (*firewall*).

Ce résumé étendu présente une vérification formelle récente [1] d'une implémentation de machine virtuelle de carte à puce réalisée par Thales à l'aide de la plateforme de vérification Frama-C [3]. Elle a été réalisée en vue d'une certification Critères Communs de niveau EAL6 (pour laquelle le certificat a été délivré en 2021). Les propriétés visées incluent les propriétés de sécurité habituelles, telles que l'intégrité et la confidentialité, qui doivent être assurées par le mécanisme de contrôle d'accès de la JCVM. Le code C de la JCVM a été annoté en utilisant le langage de spécification formelle ACSL [4].

*Cette soumission est un résumé étendu d'un article [1], publié à FM 2021.

2 Approche et contributions

Notre approche de vérification formelle de code C avec l'outil Frama-C et son greffon de vérification déductive WP [3] s'intègre dans un contexte industriel avec des exigences de sécurité très élevées.

Le code vérifié se distingue par une taille relativement importante (plus de 7.000 lignes de code C) avec des opérations bas niveau manipulant des champs de bits et des conversions de pointeurs hétérogènes. Plus de 35.000 lignes d'annotations ACSL ont été introduites directement dans le code source. La spécification en ACSL se distingue à son tour par une taille importante (5 fois la taille du code analysé) et l'expression des propriétés à prouver directement au niveau du code source. En effet, les propriétés considérées, la confidentialité et l'intégrité, sont des propriétés de haut niveau, et leur vérification déductive directement sur le code source nécessite de trouver une approche pour les prouver sous forme d'invariants globaux dans tout le code considéré. Leur spécification et vérification dans ce travail s'appuie sur MetAcsl [5], un autre greffon de Frama-C.

Dans notre publication [1], nous présentons notre approche de vérification et illustrons à travers un exemple la structure de la spécification implémentée avec un sous-ensemble de propriétés. La méthodologie que nous avons mise en place afin de satisfaire les exigences de certification Critères Communs avec des exemples d'exigences et d'annotations ACSL est présentée dans une autre publication [6].

Plus de 50.000 objectifs de preuve ont été générés par Frama-C/WP à partir du code C et des annotations ACSL associées. Plusieurs défis de passage à l'échelle ont été rencontrés et surmontés lors de ce projet afin de pouvoir prouver que le code de la JCVM respecte bien la spécification. L'utilisation de code fantôme (ou code *ghost*) a été déterminante pour trouver le bon compromis entre, d'une part, la complexité et les opérations bas niveau du code et, d'autre part, les capacités des prouveurs automatiques. Le choix précis des lemmes a permis d'augmenter le nombre d'objectifs prouvés automatiquement. Ainsi près de 99% des objectifs de preuves ont été prouvés automatiquement. Néanmoins, la création manuelle de scripts de preuve dans WP (utilisant des tactiques de preuve pour indiquer quelques premières étapes de preuve à effectuer) a été nécessaire afin de prouver le reste des objectifs.

Bien que la plupart des solutions adoptées dans notre projet ne soient pas nouvelles, leur combinaison précise a été essentielle pour mener à bout la preuve complète des propriétés exigées. Lors de ce projet, nous avons également proposé quelques extensions et améliorations de Frama-C dont certaines ont été essentielles pour le succès de ce projet. Nous présentons dans [1] les résultats de preuve détaillés, quelques leçons apprises et les améliorations souhaitées de l'outil de preuve.

3 Conclusion et travaux futurs

Ce résumé étendu présente une étude de cas de vérification formelle entièrement réalisée dans un contexte industriel en vue d'une certification. Ce travail contribue à collecter les bonnes pratiques en spécification et vérification déductive. Il établit un nouvel état de l'art des applications de la vérification déductive sur un code critique de grande taille pour la preuve de propriétés de sécurité. Les leçons tirées de ce projet ouvrent la porte à d'autres améliorations de la méthodologie et des outils de vérification déductive. Comme travaux futurs, nous prévoyons d'introduire la vérification déductive dans un processus d'intégration continue de développement logiciel.

Références

[1] A. DJOUDI, M. HÁNA et N. KOSMATOV. « Formal verification of a JavaCard virtual machine with Frama-C ». In : *the 24th Int. Symp. on Formal Methods (FM 2021)*. T. 13047. Springer, 2021, p. 427-444. DOI : 10.1007/978-3-030-90870-6_23. URL : https://nikolai-kosmatov.eu/publications/djoudi_hk_fm_2021.pdf.

[2] ORACLE. *Java Card System – Open Configuration Protection Profile, Version 3.1*. Rapp. tech. Oracle, 2020.

[3] F. KIRCHNER, N. KOSMATOV, V. PREVOSTO, J. SIGNOLES et B. YAKOBOWSKI. « Frama-C : A software analysis perspective ». In : *Formal Asp. Comput.* (2015), p. 1-37. DOI : 10.1007/s00165-014-0326-7.

[4] P. BAUDIN, P. CUOQ, J.-C. FILLIÂTRE, C. MARCHÉ, B. MONATE, Y. MOY et V. PREVOSTO. *ACSL : ANSI/ISO C Specification Language*. 2021. URL : https://www.frama-c.com/download/acsl.pdf.

[5] V. ROBLES, N. KOSMATOV, V. PREVOSTO, L. RILLING et P. L. GALL. « MetAcsl : Specification and Verification of High-Level Properties ». In : *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2019)*. T. 11427. LNCS. Springer, 2019, p. 358-364. DOI : 10.1007/978-3-030-17462-0_22.

[6] A. DJOUDI, M. HÁNA, N. KOSMATOV, M. KŘÍŽENECKÝ, F. OHAYON, P. MOUY, A. FONTAINE et D. FÉLIOT. « A Bottom-Up Formal Verification Approach for Common Criteria Certification : Application to JavaCard Virtual Machine ». In : *Proc. of the 11th European Congress on Embedded Real-Time Systems (ERTS 2022)*. Juin 2022. URL : https://nikolai-kosmatov.eu/publications/djoudi_hkkomff_erts_2022.pdf.