Sujet de thèse CIFRE

Contre-mesures logicielles "flot de données" pour la sécurité de bout-en-bout

12 mars 2026

VERIMAG (Grenoble – France) / STMicroelectronics (Grenoble – France)

Encadrement académique :

Laure Gonnord (Professeure des Universités, Grenoble-INP ÉSISAR/LCIS)
laure.gonnord@grenoble-inp.fr
Bruno Ferres (Maître de Conférences, UGA/VERIMAG)
bruno.ferres@univ-grenoble-alpes.fr

Encadrement industriel :

Yves Janin (Ingénieur, docteur en sciences)
yves.janin@st.com
François de Ferrière (Ingénieur)
francois.de-ferriere@st.com

Mots-clés : compilation ; sécurité matérielle ; contre-mesures ; analyse statique

1 Contexte

La sécurité des systèmes embarqués est devenue un sujet critique dans de très nombreux domaines tels que l'automobile, l'avionique, la défense ou l'internet des objets. Dans un modèle où l'attaquant dispose d'un accès direct au matériel, la sécurité informatique peut-être compromise dans des phases critiques (chiffrement, authentification, parties critiques d'un logiciel...) ce qui justifie le développement de protections matérielles ou logicielles afin d'atténuer les risques encourus. Typiquement, les protections implémentées visent à prévenir ou à détecter une exécution incorrecte ; leur coût en ressource matérielle, en temps d'exécution et, in fine, leur coût économique peuvent devenir conséquents.

Nous nous intéressons ici aux contre-mesures logicielles insérées dans le processus de compilation, et nous nous focaliserons sur les contremesures de type protection du flot de données. Ces contremesures consistent à dupliquer un flot de calcul en utilisant des variables en mémoire qui peuvent elles aussi être dupliquées. L'accroissement en taille de code et l'impact en performance sur le code protégé demeurent cependant très importants, de l'ordre de x3 à x4 pour la taille de code.

Une première étude [Mad25] a montré que la duplication sélective de variables bien choisies permettait une réduction importante du coût pour une baisse limitée de l'efficacité de la protection contre les attaques.

Dans cette thèse, nous proposons donc d'étudier et d'améliorer la protection du flot de données en répondant à trois objectifs :

- Sélectionner par une approche statistique les variables locales à protéger dans le but de réduire l'impact en taille de code des protections appliquées tout en garantissant un niveau de protection élevé ;
- En complément ou en remplacement de l'approche statistique précédente, donner au programmeur les moyens de définir des propriétés sur du code à protéger ;
- Limiter autant que possible l'impact de cette protection sur les optimisations effectuées par le compilateur.

Le travail proposé est à la fois une contribution à l'état de l'art des sécurisations logicielles via la compilation, en lien avec les modèles de fautes de la littérature, et une contribution méthodologique pour le support au développement de telles passes.

2 État de l'art

Modèles de faute Les attaques par injections de faute désignent toutes les attaques physiques qui modifient l'exécution nominale du système. Ces attaques sont difficiles à prendre en compte, en raison de la large variété des interférences possibles. L'impact de ces attaques est suffisamment important pour être occasionnellement visible par le grand public (par exemple, avec l'attaque RowHammer [KDK+14]).

Pour étudier ces attaques, la notion de **modèle de faute** est primordiale, car elle permet de guider finement la conception des schémas de protection. Ces modèles permettent également de désigner les zones sensibles du code à protéger, que ce soit dans un contexte d'attaques par canaux auxiliaires, ou par injection de faute. En particulier, dans cette thèse, nous nous intéresserons à la prise en compte de modèles d'attaques par injection de faute pour la génération de code protégé.

En général, les contre-mesures contre les menaces de sécurité durcissent soit le logiciel, soit le matériel, afin de garantir certaines propriétés de sécurité en présence de modèles de faute donnés. En logiciel, cette tâche consiste quasi systématiquement à modifier la chaîne de compilation, soit en ajoutant automatiquement des protections à la compilation, soit en permettant à l'utilisateur·ice de définir manuellement des protections dans le code d'origine, et en lui assurant que ces protections seront préservées lors de la compilation.

Outils Cette thèse s'articulera autour des protections de type duplication du flot de données contre les attaques par injection de fautes. Elle s'appuiera sur deux outils basés sur le compilateur LLVM, SecSwift [dFJM26] et Tracing LLVM [Mic25], ayant fait l'objet chacun de plusieurs années d'études et de recherche ; qui prennent en compte les deux approches précédentes (*sécurisation* et *préservation*).

SecSwift a été développé au sein de STMicroelectronics pour répondre à des besoins industriels et remplacer des implémentations manuelles, coûteuses en temps de développement et en maintenance, et finalement fragiles, par de la génération automatique à la compilation. L'outil SecSwift implémente la duplication du flot de données. Il implémente également une protection des variables allouées en mémoire, qui duplique tant l'espace mémoire alloué que les opérations de lecture et d'écriture pour les variables concernées. La mise en place d'une continuité entre les deux protections, protections des variables résidant en mémoire et protection du flot de calcul, est un axe de travail important.

Il s'avère cependant qu'en raison des contraintes de taille de code et de performance, la duplication systématique — telle qu'effectuée aujourd'hui — de l'intégralité d'une fonction ou même d'une sous partie, n'est pas adaptée aux enjeux industriels. Comme mentionné en introduction, une étude récente, effectuée en 2025 [Mad25], a permis d'expérimenter la duplication d'un sous-ensemble des variables d'une fonction et du flot de donnée associé et de confirmer qu'il était possible d'obtenir une réduction significative du coût de la protection tout en conservant un haut niveau de détection des attaques.

Tracing LLVM [1] est une extension de LLVM développée durant la thèse de Sébastien Michelland [Mic25] pour permettre la compilation de contre-mesures de sécurité contre des attaques par injection de fautes et par canaux auxiliaires dans LLVM. Elle réduit la friction entre le code non-fonctionnel dédié à la sécurité et la chaîne de compilation purement fonctionnelle de LLVM avec deux approches :

- L'**opacification** (technique développée durant la thèse de Son Tuan Vu [Vu21]) enrobe les primitives de sécurité de façon à ce que la préservation de la sécurité devienne une conséquence de la préservation de la sémantique ;
- Le **tracé** (technique développée durant la thèse de Sébastien Michelland [Mic25]) maintient et fournit une connexion forte entre les éléments de sécurité du programme source et ceux des programmes intermédiaires, permettant aux contre-mesures bas-niveau d'exploiter des annotations émises dans le code source.

L'outil Tracing LLVM propose, à la date d'écriture de ce sujet :

- un *front-end* pour le.a développeuse d'applications : des extensions de langage pour spécifier les variables/valeurs à protéger. Pour l'instant, il n'y a pas d'extension pour spécifier des parties de code à protéger.
- des outils *middle-end* pour le·a developpeuse de passes de sécurité : tracé des éléments précédents et surtout préservation de ceux-ci, notamment dans les passes d'optimisation. Pour l'instant, seules quelques passes d'optimisations ont été validées expérimentalement.
- des outils de préservation/log pour le *back-end*, en cours de développement, dans le cadre de la thèse de Clara Bourgeais au LCIS [BGH25].

Validation & transmission des travaux En plus des validations expérimentales du code sécurisé, les contributions logicielles feront l'objet de publications. Les contributions à Tracing LLVM seront *open source*, ainsi que certaines passes de sécurisation.

3 Contributions attendues

Nous allons détailler ci-dessous plusieurs aspects qui seront abordés au cours de cette thèse.

- Étude statistique sur le choix des variables à dupliquer.
  Comme l'a montré l'étude de 2025 [Mad25], certaines variables sont plus critiques que d'autres pour s'assurer de la bonne exécution d'un programme. Nous avons ainsi pu constater que la protection des variables impliquées dans le contrôle du nombre d'itérations des boucles ou dans le calcul de la valeur de retour d'une fonction permettent de détecter en général un nombre important de cas d'attaques. D'autres analyses résultant d'une analyse plus fine du flot de contrôle et des effets de bord d'une fonction seront examinées et implémentées dans le cadre de cette thèse. L'outil Tracing LLVM sera utilisé pour valider la robustesse de passes de sécurisation (suivi des variables tracées), et cela permettra également de *cross*-valider le système de tracé de bout en bout.
- Définition par le programmeur.se de propriétés du code à protéger.

1. Tracing LLVM est distribué sous la licence open-source Apache 2.0 utilisée par LLVM, et est accessible publiquement sur l'instance Gitlab de l'Université Grenoble–Alpes : https://gricad-gitlab.univ-grenoble-alpes.fr/tracing-llvm/

En remplacement ou en complément de l'approche statistique présentée ci-dessus, une approche guidée par le programmeur sera explorée ; celle-ci nécessitera des analyses statiques supplémentaires. Pour prendre un exemple concret, le compteur d'une boucle `for` est bien souvent remplacé au cours des optimisations effectuées par le compilateur en une ou plusieurs variables d'induction, dont la valeur initiale et le pas sont adaptées aux différents cas d'utilisations dans la boucle. Ce n'est donc pas uniquement la variable initiale que l'on peut vouloir protéger, mais bien l'ensemble des variables d'induction dérivées de cette variable initiale. De manière générale, il faut donc pouvoir exprimer des propriétés, puis les tracer dans le flot de compilation. Le travail envisagé reprend donc les travaux de la thèse de Sébastien Michelland [Mic25], en particulier son analyse de teinte sur les variables à protéger, en les adaptant à la protection de flots de données. Pour aller plus loin, il s'agira de déterminer une notion quantitative allant au-delà de protégé/non protégé, et également d'effectuer des analyses/propagations à la granularité de la fonction.

— Unification des protections du flot de données et de la mémoire.

  La protection de la mémoire implémentée par l'outil SecSwift nécessite de s'intéresser aux aspects langage de cette protection dès lors que l'on veut pouvoir manipuler des pointeurs sur des objets dupliqués. Un support partiel est disponible dans l'outil SecSwift, mais sa formalisation et sa généralisation doivent être poursuivies. En effet, un pointeur sur une variable protégée devient une structure de deux pointeurs, l'un pointant sur la variable initiale et l'autre sur la variable dupliquée. Les pointeurs sur les variables scalaires protégées ont d'ores et déjà été complétement implémentés et validés, ainsi que leur extension aux tableaux à une dimension. Mais la généralisation de cette approche doit permettre de contrôler la duplication au niveau d'un élément, d'une ligne, d'une matrice, etc. ainsi que la duplication d'une structure, qui peut elle aussi faire partie d'un tableau. Il y a donc necessité d'une réflexion au niveau du langage, afin de minimiser les modifications apportées au code source pour les variables protégées.

  La combinaison de la protection mémoire et de celle du flot de données devra également être soigneusement étudiée afin d'aboutir à une implémentation efficace en termes de génération de code et de continuité des protections. Ce sont deux domaines qui, à notre connaissance, n'ont pas fait l'objet de recherches documentées.

— Impact des protections sur l'optimisation du code.

  Dans un scénario idéal de duplication du flot de données, le code généré pour les flots principal et dupliqué devrait être identique au code généré sans duplication du flot de données. Dans la réalité, il est très difficile de ne pas perturber les passes d'optimisations du compilateur par la duplication du flot de données. L'application aussi tardive que possible de cette protection dans l'enchaînement des passes du compilateur peut être une solution pour atteindre cet objectif, et fait l'objet d'une thèse en cours au LCIS [BGH25]. Cependant, l'implémentation de cette protection plus tôt dans le processus de compilation présente également de nombreux avantages, comme nous l'expliquons dans notre article accepté à CGO [dFJM26]. Tracing LLVM permet déjà de gérer un certain nombre de cas simples, au moins jusqu'à la fin du *middle-end*.

— Validation du niveau de protection par simulation d'injection de fautes.

  STMicroelectronics utilise un outil de simulation d'injection de fautes développé en interne [CdFB21], pour évaluer le taux de résistance aux fautes des codes applicatifs. Cet outil fournit une information suffisamment fiable pour qu'il puisse être utilisé pour valider les différentes approches statistiques ou propriétés de protection qui seront expérimentées.

4 Programme

— **1ère année**

L'étudiant·e dressera une bibliographie aussi complète que possible sur les contre-mesures logicielles par duplication du flot de données. Il ou elle se familiarisera avec les outils utilisés chez STMicroelectronics (le module de contre-mesures SecSwift du compilateur LLVM) et l'outil Tracing LLVM. [1] A partir de ces connaissances, l'étudiant·e commencera une réflexion sur les différentes solutions pouvant permettre d'atteindre les objectifs visés pour cette thèse. Cette réflexion permettra de préparer les communications futures et de proposer des axes concrets pour la suite de la thèse. La première année sera l'occasion de proposer un *benchmark* complet pour la validation des méthodologies de la thèse, ainsi que des outils de mesure de la qualité des codes générés [Mad25]. Une première contribution bibliographique sera réalisée.

— **2ème année**

L'étudiant·e mettra en place, sous forme de prototype dans un premier temps, les solutions envisagées afin de vérifier concrètement qu'elles permettent de réduire significativement la taille finale du code protégé tout en conservant un haut niveau de résistance aux fautes. Il ou elle fera ensuite une implémentation permettant de passer à l'échelle, c'est-à-dire de pouvoir être utilisée sur des applications représentatives (basé sur le *benchmark* implémenté en premiere année)

— **3ème année**

L'étudiant·e sera amené à prendre du recul sur les objectifs fixés, ceux remplis, ceux qui resteront à réaliser. Il ou elle pourra rédiger un ou plusieurs articles de recherche sur les résultats obtenus, et finalisera le manuscrit de thèse.

5 Encadrement

L'étudiant·e partagera son temps entre STMicroelectronics (50%) et LCIS/Verimag (50%) afin d'obtenir tout le support, à la fois théorique et technique, nécessaire à la réalisation du projet.

STMicroelectronics STMicroelectronics a plusieurs dizaines d'années d'expérience dans le domaine de la compilation, et depuis quelques années dans l'application au domaine de la cybersécurité. Cette thèse fait suite à plusieurs stages et alternances sur les sujets des contre-mesures logicielles et de l'injection de faute par simulation. Le contexte industriel permettra de guider les choix, garantissant une valorisation rapide des résultats de cette thèse. STMicroelectronics fournira en particulier les outils de compilation et de génération de code, dont le module de sécurité SecSwift couplé au compilateur LLVM.

STMicroelectronics,
12 Rue Jules Horowitz,
38000, Grenoble

LCIS / Verimag L'encadrement académique sera composé de Laure Gonnord, Professeure des Universités à Grenoble INP - UGA et au LCIS, et Bruno Ferres, Maître de Conférences à l'Université Grenoble Alpes et à Verimag. Cette équipe d'encadrement propose une expertise théorique forte dans les domaines de la compilation (notamment les analyses nécessaires pour

la préservation des propriétés de sécurité), ainsi qu'une expertise technique sur l'outil Tracing LLVM (via l'encadrement des thèses de Sébastien Michelland et de Clara Bourgeais au LCIS).

Laboratoire VERIMAG, Bâtiment IMAG,
150 place du Torrent,
38401, Saint-Martin-d'Hères

6 Candidature

Le ou la candidat·e doit justifier d'un diplôme de M2 (ou équivalent) en informatique ou dans un domaine connexe, avec des compétences fortes en compilation. Un intérêt pour le domaine de la cyber-sécurité et/ou les aspects formels de l'informatique (notamment, la sémantique des langages) seront valorisés lors de l'évaluation des candidatures.

Pour candidater, contacter les encadrant·es en leur transmettant un CV détaillé, une courte lettre de motivation, ainsi que tout document pouvant appuyer la candidature. Après ce premier contact, il conviendra de candidater en ligne à l'offre suivante : https://stmicroelectronics.eightfold.ai/careers/job/563637170403723.

Références

[BGH25] Clara Bourgeais, Laure Gonnord, and David Hély. Work in Progress : Securing a Critical Variable at Compile Time. In *Colloque 2025 du GDR SOC2*, Lorient (56100), France, June 2025.

[CdFB21] Hervé Chauvet, Francois de Ferrière, and Thomas Bizet. Sofware Fault Injection for SecSwift qualification. Journée thématique sur les attaques par injection de fautes (JAIF), Paris, France, 2021. https://jaif.io/2021/programme.html.

[dFJM26] Francois de Ferriere, Yves Janin, and Sirine Mechmech. SecSwift, a Compiler-Based Framework for Software Countermeasures in Cybersecurity. In *2026 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, Los Alamitos, CA, USA, February 2026. IEEE Computer Society.

[KDK+14] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping bits in memory without accessing them : An experimental study of dram disturbance errors. *ACM SIGARCH Computer Architecture News*, 42(3) :361–372, 2014.

[Mad25] Rozeanne Madondo. Memory Protection in a Cybersecurity Module of the LLVM Compiler. Master's thesis, Université Grenoble Alpes, France, 2025.

[Mic25] Sébastien Michelland. *Compilation pour la sécurité matérielle : au-delà de la sémantique*. PhD thesis, Université Grenoble Alpes, 2025.

[Vu21] Son Tuan Vu. *Optimizing Property-Preserving Compilation*. PhD thesis, Sorbonne University, 2021.