AUTEURS

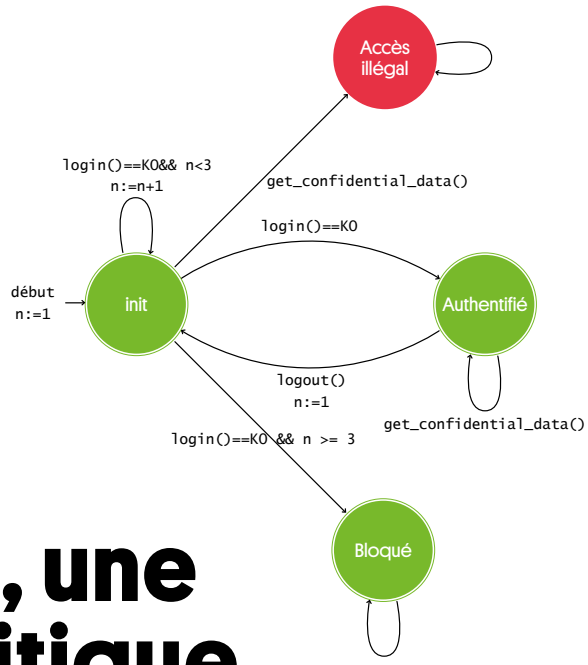
Sébastien Bardin
[Direction de la recherche technologique]
Ingénieur-chercheur au Département d'ingénierie des logiciels et des systèmes du CEA-List.

Lesly-Ann Daniel
[Direction de la recherche technologique]
Doctorante au Département d'ingénierie des logiciels et des systèmes du CEA-List.

Virgile Prévosto
[Direction de la recherche technologique]
Ingénieur-chercheur au Département d'ingénierie des logiciels et des systèmes du CEA-List.

Julien Signoles
[Direction de la recherche technologique]
Ingénieur-chercheur au Département d'ingénierie des logiciels et des systèmes du CEA-List.

Patrick Tessier
[Direction de la recherche technologique]
Ingénieur-chercheur au Département d'ingénierie des logiciels et des systèmes du CEA-List.



→
Figure 1 :
exemple d'automate.

Le temps, une notion critique en développement logiciel

— On aurait pu croire que l'accélération des performances des processeurs, capables de réaliser plusieurs milliards d'opérations par seconde, allait résoudre les problèmes temporels des systèmes informatiques (smartphones, programmes de bureau, etc.). Il n'en est rien. Car, au-delà de la vitesse pure, il est notamment important d'ordonnancer correctement les opérations à réaliser pour que chacune intervienne au moment adéquat, sans perturber le fonctionnement global du système. Dans certains cas, il peut même être utile de modéliser mathématiquement cette temporalité afin de démontrer que toutes les actions s'enchaînent parfaitement. En outre, en cybersécurité, le temps est un vecteur d'attaque exploitable pour découvrir des informations confidentielles. Il convient dès lors d'en protéger le système.

Aujourd'hui, tout système informatique effectue plusieurs actions simultanément. Ainsi, votre smartphone vous affiche des informations tout en réagissant aux pressions de vos doigts sur son écran, envoie et reçoit des données *via* différents réseaux (téléphone, Bluetooth, 4G, Wifi) et gère l'ensemble des applications que vous avez ouvertes. La rapidité des calculs ne suffit pas à assurer la simultanéité des opérations et l'immédiateté des réponses : si votre smartphone n'était que rapide, il pourrait, par exemple, passer continuellement son temps à vérifier l'arrivée de nouveaux messages sans jamais réagir à vos pressions sur l'écran. Pour que chaque tâche soit réalisée au bon moment, d'une manière fluide, il faut ordonnancer correctement les opérations composant chaque tâche.

Ordonnancer pour optimiser

Mais trouver le bon ordonnancement est compliqué car il faut prendre en compte de multiples contraintes : vitesse des opérations, temps pour passer d'une tâche à une autre, réaction rapide à un événement extérieur (comme la voix) survenant après une attente arbitrairement longue, impossibilité de partager certaines ressources entre opérations (deux tâches ne peuvent écrire en même temps dans le même fichier). Un bon algorithme d'ordonnancement doit donc notamment garantir que chaque tâche accède équitablement aux ressources du système et que ces dernières sont utilisées au maximum de leur capacité : il ne serait, par exemple, pas efficace de laisser un processeur au repos alors qu'il y a des calculs en attente. D'autres propriétés peuvent être aussi attendues en fonction du contexte. Ainsi, un avion doit notamment réagir rapidement aux actions du pilote et éviter de ne pas recevoir les valeurs de son altimètre pendant trop longtemps.

Logique temporelle et vérification de programmes

Lorsque des vies sont en jeu ou que des dégâts environnementaux et/ou économiques sont possibles, pour s'assurer que les programmes exécutés satisfont bien les propriétés attendues, il faut les modéliser mathématiquement. Pour les propriétés reposant sur une notion de temps, la logique mathématique classique ne suffit pas : on a recours à des *logiques temporelles* pour caractériser des séquences d'événements qui surviennent durant l'exécution d'un programme.

Par exemple, on peut vouloir vérifier que tout appel à une fonction traitant des données confidentielles est précédé par un appel réussi à une fonction connectant et identifiant l'utilisateur de ces données, sans qu'aucun appel à une fonction de déconnexion n'ait eu lieu dans l'intervalle. Par ailleurs, trois échecs de connexion consécutifs doivent conduire à une erreur. Ces propriétés peuvent être exprimées soit comme des formules logiques utilisant des opérateurs spécifiques pour parler des événements futurs et/ou passés, soit sous forme d'*automate* précisant les différents états possibles du système et comment on passe de l'un à l'autre. On vérifie ensuite que toute exécution du programme modélisé par l'automate satisfait bien les propriétés souhaitées, en particulier le fait que l'état illégal n'est jamais atteint **(figure 1)**.

Le temps-constant pour la cybersécurité

Au-delà du bon fonctionnement du système en situation nominale (sûreté), il faut également s'assurer de sa sécurité, c'est-à-dire du fait que rien de grave ne peut survenir en présence d'un attaquant. Aujourd'hui, un pirate est capable d'utiliser le temps à son avantage. C'est le cryptographe américain Paul Kocher qui a décrit, le premier, en 1996, une *attaque temporelle* consistant à découvrir une clé cryptographique secrète en se basant sur le fait que les calculs effectués pour chacun des bits (0 ou 1) de la clé étaient plus nombreux lorsque le bit valait 1 : cette différence de temps d'exécution est mesurable et permet de retrouver l'intégralité de la clé secrète par analyse statistique.

Le temps d'exécution dépend aussi des accès à la mémoire. Par exemple, les données lues fréquemment par un programme sont automatiquement stockées dans une petite mémoire très rapide d'accès, le cache : les *attaques par cache* exploitent les différences de temps pour en extraire des informations **(figure 2)**. Le chiffrement AES, très utilisé aujourd'hui, est vulnérable à ce type d'attaque. La discipline de programmation *temps-constant* permet de se protéger de ces attaques temporelles en écrivant le programme de manière à ce que les instructions exécutées et les accès mémoires soient indépendants des secrets. Cette contre-mesure est appliquée dans un grand nombre de programmes cryptographiques que nous utilisons quotidiennement, par exemple pour chiffrer nos communications ou naviguer sur Internet. Vérifier que ces contre-mesures sont réellement efficaces est un sujet de recherche actif.

Au CEA-List, le Département d'ingénierie des logiciels et des systèmes (DILS) développe plusieurs logiciels intégrant différentes notions de temporalité dans leurs modélisations et leurs analyses : la suite d'outils de modélisation de systèmes Papyrus, la plateforme d'analyse de programme Frama-C et l'analyseur pour la cybersécurité Binsec. ●

« Au CEA-List, le Département d'ingénierie des logiciels et des systèmes développe plusieurs logiciels intégrant différentes notions de temporalité dans leurs modélisations et leurs analyses. »

Figure 2 : exemple d'attaque par cache.