Sécurité des systèmes industriels : filtrage applicatif et recherche de scénarios d'attaques

Maxime Puys

Communauté
UNIVERSITÉ Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE LA COMMUNAUTÉ UNIVERSITÉ GRENOBLE ALPES

Spécialité : **Informatique**

Arrêté ministérial : 25 mai 2016

Présentée par

Maxime Puys

Thèse dirigée par **Marie-Laure Potet**

préparée au sein du laboratoire **Vérimag**
et de **École Doctorale de Mathématiques, Sciences et Technologies de l'Information, Informatique**

Sécurité des systèmes industriels : filtrage applicatif et recherche de scénarios d'attaques

Thèse soutenue publiquement le **5 février 2018**,
devant le jury composé de :

M. Jean-Marie Flaus
Professeur à l'Université Grenoble Alpes, Président
M. Hervé Debar
Professeur à Télécom SudParis, Rapporteur
M. Jérémie Guiochet
Maître de conférences à l'Université de Toulouse III, Rapporteur
M. Frédéric Dadeau
Maître de conférences à l'Université de Franche-Comté, Examinateur
M*me* Marie-Laure Potet
Professeur à Grenoble INP, Directrice de thèse
M. Jean-Louis Roch
Maître de conférences à Grenoble INP, Encadrant

Remerciements

> Il faut trois conditions pour faire le thé : le temps, les braises et les amis.
>
> *(Proverbe touareg)*

Comme le thé dans le proverbe, cette thèse a nécessité du temps et de l'énergie mais surtout de nombreuses personnes sans qui rien n'aurait été possible. Aussi je souhaite remercier tout particulièrement **Marie-Laure Potet**, pour la qualité de son encadrement tout au long de ma thèse, et pour avoir cru en ma capacité à mener à bien ce projet. Elle m'a beaucoup apporté tant sur le plan professionnel que personnel. Je remercie également **Jean-Louis Roch**, pour son encadrement, ses capacités de médiation, et sa créativité. À tous les deux, merci pour votre bonne humeur sans faille, c'était un plaisir de travailler avec vous.

Merci à **Jérémie Guiochet** et **Hervé Debar**, pour avoir accepté de rapporter sur ma thèse. Merci également à **Jean-Marie Flaus** et **Frédéric Dadeau**, pour avoir accepté de participer à mon jury. Vos remarques et vos questions pertinentes ont été une source d'inspiration pour améliorer le présent manuscrit et ouvrent la voie à des travaux futurs.

Merci à **Pascal Lafourcade**, pour avoir supervisé une grande partie de mes travaux comme pour l'un de ses propres doctorants. Merci également à **Jean-Marc Vincent**, pour ses enseignements lors des cours de magistère. À tous les deux, votre rigueur scientifique m'a guidée sur cette voie et restera un exemple à mes yeux.

Merci à **Laurent Mounier**, **Jannik Dreier**, et **Jean-Guillaume Dumas**, pour leurs contributions et leurs conseils sur les différents aspects de cette thèse. Merci de façon plus général à mes professeurs d'informatique et plus particulièrement de sécurité pour m'avoir offert une vision aussi large que possible du domaine, plusieurs d'entre eux sont cités ci-dessus. J'espère avoir à nouveau l'occasion de travailler avec vous à l'avenir.

Merci au consortium du projet ARAMIS qui a financé ma thèse. Nos échanges, souvent cordiaux, parfois tendus, ont largement contribué à me faire monter en compétence sur les spécificités des systèmes industriels, qui sont au cœur de cette thèse. Cette collaboration m'a également donné une vision de la recherche dans l'industrie. Merci également aux doctorants du projet, **Jean-Baptiste Orfila**, **Emmanuel Perrier**, et **Nicolas Kox** qui ont partagé mon sort pendant ces trois ans.

Merci à mes collègues du laboratoire qui m'ont permis de décompresser pendant ces trois ans. À **Hamza**, **Josselin**, **Louis**, **Cristina**, **Dinh**, **Sahar**, **Yuliia**, et **Valentin**,

pour les pauses déjeuner animées et pour nos discussions. Aux joueurs de coinche **Denis**, **Amaury**, **Alexandre**, et **Alexis**, pour leur niveau de jeu :-) À mes amis **Pierre**, **Carole-Anne**, **Alexandre**, **Typhaine**, **Julien**, **Valentin**, et **Benjamin**, pour m'avoir aidé à lever les yeux de mon écran et à sortir de chez moi.

Je voudrais adresser un remerciement tout particulier à **Anaïs**, pour sa patience, pour ses multiples conseils et pour m'avoir toujours poussé, sans s'en rendre compte, à donner le meilleur de moi-même. Ces années à tes côtés m'ont aidé à dépasser les moments parfois difficiles qui ont pu apparaître ces derniers temps.

Enfin, je voudrais remercier mon père **Philippe**, ma mère **Fabienne** et mon frère **Valentin** pour tout ce qu'ils font pour moi depuis toujours. Je ne serais pas le quart de ce que je suis sans l'aide et l'amour que vous m'apportez tous les jours.

Table des matières

Première partie

Introduction

Chapitre 1

Contexte et motivations

Beware of programmers who carry screwdrivers.

(Leonard Brandwein)

Résumé du chapitre

Les systèmes industriels sont des moyens informatiques contrôlant et pilotant un procédé industriel. Ces « procédés » désignent par exemple la production et la distribution d'électricité, le traitement des eaux, ou encore les transports. Il apparaît depuis plusieurs années que les systèmes industriels sont vulnérables à des attaques informatiques. Cela peut s'expliquer car ils n'ont pas été conçus avec des contraintes de sécurité notamment de part leur isolation physique des réseaux non sûrs tels qu'Internet. Cependant, ils doivent aujourd'hui faire face à une multitude d'attaquants avec des objectifs et des capacités variés. Dans ce chapitre, nous décrivons succinctement les spécificités des systèmes industriels, puis présentons une sélection de normes et bonnes pratiques en terme de sécurité pour les systèmes industriels. Enfin nous montrons différents types d'attaquants pouvant viser les systèmes industriels.

Sommaire

1.1. Les systèmes industriels

Les systèmes industriels, souvent appelés *Industrial Control Systems* (ICS), ou encore SCADA (pour Système d'acquisition et de contrôle de données – *Supervisory Control And Data Acquisition*) par abus de langage, sont des moyens informatiques contrôlant et pilotant un procédé industriel. Historiquement, ces « procédés » ont désigné une multitude de dispositifs présents au cœur de nos vies. On pense par exemple facilement aux centrales nucléaires ou aux barrages, mais aussi à n'importe quelle usine (par exemple de production alimentaire). De nos jours, l'augmentation des interconnexions et l'arrivée de l'Internet des objets industriels – *Industrial Internet of Things* (IIOT) – étend les systèmes industriels à des objets avec lesquels nous interagissons tous les jours. Ainsi, les compteurs électriques, les ascenseurs, les feux de signalisation ou encore les trains peuvent maintenant être connectés via Internet. On désigne aussi souvent cette transition comme l'*Industrie 4.0*. De nombreux spécialistes considèrent que l'Industrie 4.0 et l'IIOT désignent globalement le même domaine vu par deux communautés historiquement disjointes. On distingue d'ailleurs souvent les systèmes industriels en temps que OT (pour *Operational Technology*) des systèmes d'information traditionnels dits IT (pour *Information Technology*). Hors, ce fossé entre les automaticiens et les informaticiens est d'autant plus dommageable que ces deux communautés sont aujourd'hui menées à collaborer sur des problématiques communes et le besoin de personnes capables de faire interface entre les deux mondes est crucial.

Dans le cadre de cette thèse, nous nous intéressons à la sécurité des systèmes industriels. En effet, ces derniers régissent aujourd'hui des aspects essentiels de nos vies tels que la production et la distribution d'électricité, le traitement des eaux, la santé ou encore les transports. En plus de leur criticité, la question de la sécurité de ces systèmes se pose car ils n'ont pas été conçus dans ce but, notamment de part leur isolation physique des réseaux non sûrs tels qu'Internet et leur besoin vital de rentabilité. Cependant, ils doivent aujourd'hui faire face à une multitude d'attaquants allant aussi bien des terroristes qu'à des employés mécontents en passant par des états, des mafias, des *hacktivistes* ou encore des *script kiddies*. Ces attaques peuvent par ailleurs avoir des objectifs très variés tels que l'espionnage industriel, l'atteinte à l'image ou le sabotage. Ce besoin de sécurité rend le sujet sensible et de nombreuses lois, normes et guides de bonnes pratiques ont été rédigés par des consortiums industriels et des agences gouvernementales. En France, le directeur général de l'Agence Nationale de la Sécurité des Systèmes d'Information (ANSSI), Guillaume Poupard, a notamment déclaré en juin 2017 que la sécurité des systèmes industriels est désormais la « priorité des priorités » de l'Agence [1].

Les systèmes industriels ne sont pas pour autant démunis de toutes protections. De très nombreux travaux ont été entrepris dans le domaine de la sûreté de fonctionnement. Cette thématique vise à garantir que le système se maintiendra dans un état stable et résistera à des pannes ou des erreurs. Cependant, elle ne considère pas la possibilité qu'un attaquant cherche activement à porter atteinte au système. A l'opposé, beaucoup de travaux en sécurité s'intéressent aux méthodes qu'un attaquant pourrait mettre en œuvre afin de prendre le contrôle de systèmes (industriels ou non) tels que les *buffer-overflow*, le *cross-site scripting*, ou encore la cryptographie faible. La question que nous nous po-

1. https://fr.reuters.com/article/technologyNews/idFRKBN18X1ZF-OFRIN

sons dans nos travaux est la suivante : supposons qu'un attaquant ait pris le contrôle d'une partie du système, qu'est-il en mesure de faire par la suite ? Pour cela, nous nous intéresserons essentiellement à des attaques ciblées portant sur les communications au niveau applicatif entre des équipements. Nous considérons ainsi des attaquants menant des attaques ciblées et pouvant potentiellement avoir des connaissances sur l'architecture du système, sur sa logique applicative voir corrompre une partie des équipements. De telles attaques sont souvent désignées comme « menaces persistantes et avancées » ou *Advanced Persistent Threat* (APT). Cette thèse a été financée par le projet PIA ARAMIS (P3342-146798) visant à développer un dispositif de rupture de protocoles et de filtrage de communications industrielles.

Plan de la thèse : La suite du chapitre 1 décrit plus en détail les spécificités des systèmes industriels, puis présente une sélection de normes et guides en terme de sécurité pour les systèmes industriels. Il liste ensuite différents types d'attaquants et motive les contributions de cette thèse en se basant sur des attaques ayant eu lieu. Le chapitre 2 présentera des informations techniques sur les systèmes industriels qui seront utilisées tout au long du manuscrit puis énoncera les objectifs de la thèse. Ensuite, trois chapitres de contribution seront présentés. Le chapitre 3 s'intéressera à la problématique du filtrage applicatif et montrera comment il est possible de prendre en compte des contraintes de sûreté dans les règles d'un filtre. Le chapitre 4 aura pour thème la sécurité apportée par les protocoles de communication industriels afin de vérifier formellement les propriétés de sécurité qu'ils fournissent. Le chapitre 5 proposera une méthode de propriétés de sûreté en présence d'attaquants dans des systèmes industriels. Ces trois chapitres de contribution seront articulés autour d'un fil rouge en montrant leurs apports possibles à une des attaques : l'attaque Maroochy Shire. Enfin le chapitre 6 conclura ce manuscrit en présentant un bilan des travaux effectués et des perspectives pour le domaine.

1.1.1. Particularités des systèmes industriels

Les systèmes industriels sont depuis longtemps représentés par le modèle dit de Purdue [Williams, 1991], aussi appelé *pyramide du CIM (Computer Integrated Manufacturing)* [Waldner, 1990]. Cette représentation sous forme de pyramide cherche à diviser l'architecture des systèmes industriels en strates (ou couches, ou niveaux). Le bas de la pyramide est en général le niveau le plus proche du procédé et les niveaux les plus hauts plus proches de la gestion de l'entreprise (stocks, commandes, effectifs, etc.).

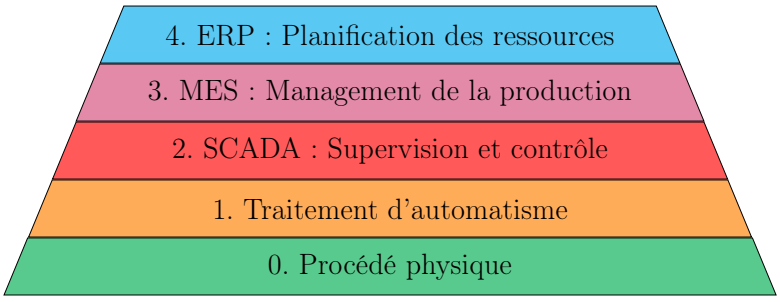


FIGURE 1.1. – Représentation de Purdue [Williams, 1991, Waldner, 1990]

La représentation communément admise est présentée en figure 1.1 et introduit les niveaux suivants :

Niveau 0 - Procédé physique : Ce niveau est constitué de capteurs et d'actionneurs interagissant avec le procédé physique contrôlé par le système industriel. Les capteurs effectuent des mesures (tension, pression, température, etc.) et les convertissent en signaux électriques qui sont envoyés au niveau 1. Les actionneurs reçoivent des signaux électriques du niveau 1 et agissent sur le procédé via des pompes, des moteurs, etc.

Niveau 1 - Traitement d'automatisme : C'est à ce niveau que se trouve la logique du système industriel. Des automates programmables pilotent les actionneurs du niveau 0 en fonction des mesures remontées par les capteurs. Ils font par ailleurs remonter des informations sur le procédé au niveau 2 qui peut pour sa part donner des ordres généraux pour réorienter le procédé. Par exemple, le niveau 2 pourra demander le démarrage ou l'arrêt d'une chaudière tandis que le niveau 1 effectuera cet ordre tout en s'assurant qu'il ne provoque pas de catastrophe (par exemple une explosion de la chaudière). Ces automates programmables peuvent prendre plusieurs formes suivant leurs caractéristiques, le contexte dans lequel ils sont utilisés et leurs dénominations sont parfois mélangés par abus de langage :

— PLC - *Programmable Logical Controler* ;
— RTU - *Remote Terminal Unit* ;
— DCS - *Distributed Control System* ;
— PAC - *Programmable Automation Controler* ;
— IED - *Intelligent Electronic Device*.

A l'origine, les PLC étaient plus utilisés pour le contrôle du procédé et les RTU pour la télémesure (technique de mesure à distance, à ne pas confondre avec télémétrie, technique de mesure des distances). Pour cela les RTU intégraient des communications sans fil. DCS, PAC et IED désignent des équipements plus récents, intégrant souvent à la fois les fonctionnalités de PLC et de RTU. On utilisera plutôt le terme *automate programmable* par la suite car plus général.

Niveau 2 - SCADA : Supervision et contrôle : Par abus de langage les systèmes industriels sont souvent appelés SCADA. Cependant le SCADA désigne plutôt la supervision du procédé. Pour cela il utilise les informations remontées par les automates programmables du niveau 1 qu'il représente de façon graphique (synoptique de circuit, courbes, etc.) aux opérateurs qui peuvent contrôler que l'ensemble du procédé reste stable. En cas de déviation, des alarmes sont affichées aux opérateurs qui peuvent alors agir sur le procédé en utilisant le SCADA pour envoyer des ordres aux automates.

Niveau 3 - MES : Management de la production : Les MES (*Manufacturing Execution Systems*) reçoivent des informations du niveau 2 et s'en servent pour mesurer des indicateurs d'analyse du système industriel (contrôle de la qualité, suivi de production, maintenance, etc.). Ils sont aussi responsables du stockage à long terme des données de l'installation.

Niveau 4 - ERP : Planification des ressources : Les ERP (*Enterprise Resource Planning*) servent à gérer les ressources de l'entreprise. Cela inclut la gestion des commandes, des stocks, de la paie, de la comptabilité, etc. Cette planification des ressources se base sur les informations remontées par le niveau 3. Les niveaux 0, 1 et 2 sont le plus souvent isolés physiquement des niveaux 3 et 4 pour des raisons de sûreté de fonctionnement.

Historiquement, chaque niveau ne communique qu'avec les niveaux du dessus et du dessous. Ces communications varient autant dans leurs formes que dans leurs fonctionnalités, en particulier :

Entre les niveaux 0 et 1 : Ces communications se font sur la base de protocoles temps réel propriétaires, souvent dépendants du domaine (par exemple MODBUS [MODBUS, 2004], CAN bus [ISO-8802, 1998], Profibus [DIN-19245, 1991], BACnet [Bushby, 1997], etc.). Dans des contextes spécifiques (tels que l'utilisation de RTU), ces protocoles peuvent accepter des latences allant jusqu'à 100 ms. Ces communications sont standardisées notamment par la norme IEC/CEI 61158 [IEC-61158, 2014], décrivant les bus de terrains.

Entre les niveaux 1 et 2 : Les protocoles ne sont plus temps réel et peuvent accuser des latences de 500 ms. On y trouve entre autres les protocoles OPC-UA [IEC-62541, 2015], MODBUS [MODBUS, 2004], S7 [Nardella, 2016], DNP3 [Clarke et al., 2004].

Entre les niveaux 2 et 3 et 3 et 4 : On y retrouve le protocole OPC-UA utilisé entre les niveaux 1 et 2 auquel viennent s'ajouter des protocoles grand public tels que FTP [Postel and Reynolds, 1985] et FTPS [Ford-Hutchinson, 2005] pour du transfert de fichiers ou encore HTTPS [Rescorla, 2000]. Dû au fait que les niveaux 3 et 4 sont isolés des niveaux 0, 1 et 2 (et donc supposément dans un environnement moins sûr), les communications entre ces niveaux passent souvent par des protocoles sécurisés. Les démarcations entre les niveaux 3 et 4 et les communications sont par ailleurs normalisées dans la norme IEC/CEI 62264 [IEC-62264, 2013]. On y retrouve par exemple des modèles abstraits (par exemple UML) des objets manipulés par ces deux niveaux et une description des interfaces permettant les communications (leur nature, leur position, etc). Ici la latence maximale s'allonge à 30 s.

Les communications entre les différents niveaux ont également la particularité d'être en forme d'arbre. Cela signifie qu'un composant de niveau N parlera souvent avec plusieurs composants de niveau $N-1$ (par exemple un SCADA rassemblera les données de plusieurs automates programmables lesquels contrôleront chacun plusieurs capteurs et actionneurs). Comme décrit dans [Allot, 2014], ce modèle pyramidal s'avère de plus en plus éloigné de la réalité du terrain. C'est notamment dû au fait que les communications ne se font plus seulement entre niveaux voisins. On trouve par exemple de plus en plus régulièrement des MES qui communiquent directement avec les procédés, en plus des informations qui leur sont remontées par les SCADA. Par exemple afin d'obtenir des informations sur l'usure du matériel et planifier les maintenances. De la même façon, les équipements deviennent de plus en plus intelligents et autonomes, ce qui estompe les différences entre les niveaux. En particulier, MES et SCADA offrent en général des

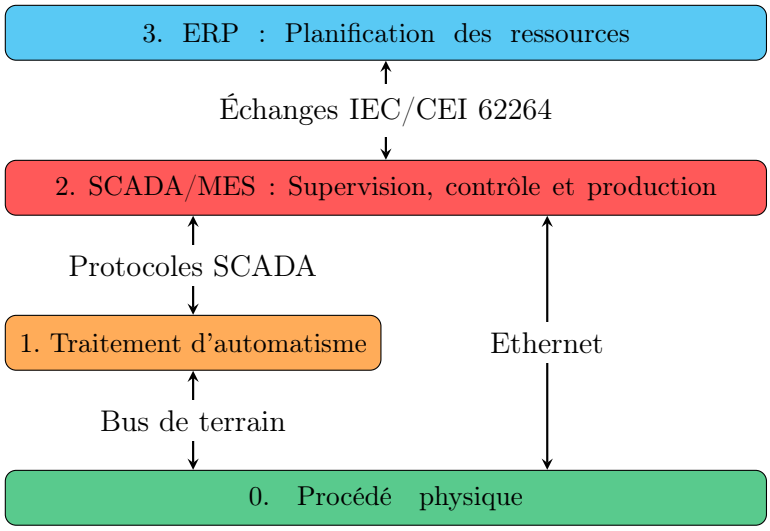


FIGURE 1.2. – SCADA et MES peuvent fusionner (vision simplifiée) [Allot, 2014]

fonctionnalités très similaires, le MES proposant une présentation plus « informatique » des données tandis que le SCADA aurait une vue plus « automatisme ». Ainsi, encore selon [Allot, 2014], MES et SCADA pourraient même fusionner et donner le modèle en figure 1.2. Dans ce modèle, le niveau 2 échange par exemple avec le niveau 1 pour contrôler le procédé et avec le niveau 0 dans des cas spécifiques pour obtenir des informations relatives à la production (météo, caméra, ordonnancement des tâches, etc.). Enfin, les SCADA sont de plus en plus connectés à des réseaux publics tels qu'Internet et utilisent de plus en plus des connexions sans fil, facilitant encore les communications inter-niveaux. On peut par exemple imaginer le cas d'un opérateur qui pourrait intervenir depuis son domicile en télémaintenance. Cet opérateur serait alors en mesure de communiquer avec des périphériques de niveau 1 ou 2 alors qu'il n'est même pas présent sur le site de l'entreprise.

1.1.2. Propriétés de sécurité requises par les systèmes industriels

Les systèmes industriels sont spécifiques de part les propriétés qu'ils requièrent. La plupart des systèmes se focalisent sur les propriétés mises en avant par la série de normes ISO 27000 [ISO-27000, 2005] (confidentialité, intégrité et disponibilité, « CIA »). Les systèmes industriels s'inscrivent en partie dans cette tendance en se concentrant sur la disponibilité et l'intégrité ; l'authenticité étant également souvent requise. A l'inverse, la confidentialité est souvent ignorée voire évitée. Plus généralement, les propriétés désirées pour un système industriel sont les suivantes.

Disponibilité : La disponibilité garantit l'absence de panne dans un système pendant les plages d'utilisation prévues (potentiellement tout le temps). En effet, comme décrit dans l'introduction, les systèmes industriels requièrent en général avant tout d'être rentables et toute interruption de service est le plus souvent un manque à gagner pour l'exploitant. Il va sans dire que dans le cas des systèmes industriels dits *critiques* (par

exemple la production d'énergie ou l'assainissement des eaux), l'arrêt de fonctionnement peut avoir de graves conséquences sur les humains et l'environnement. La disponibilité est par ailleurs une propriété dite de *vivacité*, assurant qu'une propriété sera vraie à partir d'une certaine étape de l'exécution. Dans le cas des systèmes industriels, la disponibilité vise donc également à garantir qu'une entité demandant une ressource finira par l'obtenir (par exemple un opérateur qui demande la valeur d'une variable).

Authenticité : L'authenticité, souvent appelée *authentification* par abus de langage, vise à garantir qu'une entité (un humain, un composant) est bien celui qu'il prétend. La méthode pour garantir l'authenticité peut se faire sur (i) ce que l'on sait (un mot de passe), (ii) ce que l'on possède (un certificat, un jeton d'authentification), (iii) ce que l'on est (biométrie), ou encore (iv) une information sur le contexte (par exemple où l'on se trouve). Cette méthode permettant de garantir l'authenticité est appelée authentification. Une authentification reposant sur plusieurs facteurs décrits ci-dessus est dite *forte*. Cette propriété rend notamment possible le contrôle d'accès qui permet d'éviter qu'une entité puisse effectuer des actions qui ne lui sont pas autorisées. Elle est aussi particulièrement importante dans les systèmes industriels car chaque action dans le monde logique a une conséquence - potentiellement destructrice - dans le monde physique. Il faut donc garantir que seules les entités habilitées peuvent envoyer des commandes dans la limite de leur autorisation. Le terme authentification sera utilisé par la suite car plus naturel.

Intégrité : L'intégrité est la préservation et l'assurance de la cohérence d'une donnée dans le temps. Cette définition peut beaucoup varier suivant le contexte. En particulier deux sens apparaissent dans le cas de l'intégrité de messages transférés sur un réseau. D'une part la protection contre les changements accidentels peut être garantie en utilisant notamment des mécanismes de détection d'erreur tels que les contrôles de redondance cyclique ou CRC. C'est généralement le sens que l'intégrité prendra pour un automaticien. D'autre part, l'intégrité au sens cryptographique vise à protéger la donnée contre des modifications volontaires par un attaquant. Elle requiert l'utilisation de primitives cryptographiques telles que des fonctions de hachage ou des signatures. Bien que protégeant également contre les modifications accidentelles, l'intégrité cryptographique est plus coûteuse en temps et en taille de message. C'est en partie pour cette raison que les systèmes industriels se sont historiquement basés sur la protection contre les erreurs. Qui plus est, leur isolation physique du monde extérieur leur procurait déjà une forme de protection suffisante pour se consacrer sur la protection contre les erreurs et les pannes.

Non-répudiation : La non-répudiation est l'incapacité pour un agent de nier qu'il a effectué une action ou qu'une action a été faite à son encontre. Par exemple l'incapacité pour un opérateur de nier qu'il a envoyé un message si celui-ci cause un problème. Ce pourrait aussi être l'impossibilité pour l'opérateur de nier qu'il a reçu une information sur le procédé. Cette propriété repose essentiellement sur les protocoles de communication et est donc actuellement très peu présente (voir totalement absente) dans les systèmes industriels.

Sûreté de fonctionnement : L'authentification, l'intégrité, la confidentialité et la non-répudiation sont des propriétés dites de *sûreté*. Elles ont la particularité d'être vérifiables sur une trace donnée (par opposition aux propriétés de vivacité comme la disponibilité [Alpern and Schneider, 1987]). Néanmoins la sûreté ne se limite pas aux vérifications de sécurité, en particulier la sûreté de fonctionnement vise à garantir des propriétés spécifiques à la logique applicative des systèmes. Par exemple, vérifier qu'un four ne dépasse jamais une certaine température ou encore qu'un bec verseur ne s'ouvre que lorsqu'un récipient est bien dessous. Ce sont traditionnellement des propriétés de ce type que vérifient déjà par construction les systèmes industriels. Cependant elles ne sont peu ou pas vérifiées en présence d'un attaquant. La sûreté de fonctionnement est par ailleurs souvent opposée à la sécurité et il est parfois compliqué de concilier les deux. Plusieurs travaux existent sur ces relations ; par exemple la thèse de Piètre-Cambacédès en 2010 [Piètre-Cambacédès, 2010] (qui précise comment sûreté et sécurité peuvent parfois être antagonistes ou complémentaires) ou encore la taxonomie de Avizienis *et al.* en 2004 [Avizienis et al., 2004].

Traçabilité : La traçabilité assure que les actions ou les tentatives d'actions dans le système sont conservées dans un journal. Les traces doivent aussi être exploitables en fournissant par exemple les raisons pour lesquelles une action est refusée.

Certaines propriétés de sécurité sont plus rarement requises pour les systèmes industriels et interviennent surtout lorsque des données de clients sont présentes (par exemple des relevés de compteur électrique), à savoir :

Confidentialité : La confidentialité garantit que seules les personnes autorisées ont accès aux informations qui leur sont destinées. Cette propriété est plus rare dans les systèmes industriels du fait que, comme l'intégrité, elle requiert l'utilisation de primitives cryptographiques qui sont coûteuses en temps de calcul. Par ailleurs, elle requiert que tous les éléments analysant l'état du procédé physique (sondes, systèmes de détection d'intrusions) soient capables d'accéder au contenu des messages échangés ce qui peut être compliqué lorsqu'ils sont chiffrés. La confidentialité commence néanmoins à être une propriété nécessaire par exemple lors de l'envoi de mots de passe (introduits par les protocoles de communication récents) et de données relatives à des clients sur le réseau.

Anonymisation : L'anonymisation consiste à modifier une donnée afin d'empêcher tout lien avec son possesseur. Cette propriété est spécifiquement requise en présence de données clients (et à condition que ce lien ne soit pas utile pour le système industriel). Historiquement, peu de systèmes industriels avaient à manipuler des données clients, rendant cette propriété marginale. Cependant dû à l'augmentation de l'interconnexion des systèmes et la méfiance grandissante des usagers à l'égard des collectes de données, l'anonymisation devient une propriété importante dans les systèmes industriels. Elle est par ailleurs de plus en plus mise en avant voir imposée par les réglementations, comme par exemple le règlement européen sur la protection des données personnelles applicable à partir du 25 mai 2018.

1.2. Un sujet d'actualité

La prise de conscience de l'importance de la sécurité des systèmes industriels date de presque vingt ans. En effet, dans une note de 1998 [Clinton, 1998], le président américain William – Bill – Clinton, ordonne la mise en place de protections des infrastructures critiques des États-Unis contre les attaques informatiques. Depuis cette décision, le domaine est progressivement devenu l'un des principaux théâtres d'affrontement entre les nations et de nombreux standards, normes et guides ont fait leur apparition. Plusieurs travaux de recherche se sont attelés à lister ces documents de façon aussi exhaustive que possible. On peut notamment citer Piètre-Cambacédès et Chaudet en 2010 [Piètre–Cambacédès and Chaudet, 2010] qui proposent le cadre référentiel SEMA, une méthode d'analyse documentaire pour aider à distinguer les notions de sécurité et sûreté de fonctionnement. Ce faisant, ils détaillent une importante liste de documents sur la sécurité et la sûreté dans le systèmes industriels pour illustrer leur propos. Des travaux similaires ont été entrepris par l'auteur du blog *Secur'ID*[2] en 2011 et poursuivis par le CLUSIF[3] en 2014. Une collection de ces références est enfin régulièrement mise à jour sur le site SCADAhacker[4]. Nous présentons ici une sélection de documents considérés comme emblématiques dans le domaine de la sécurité des systèmes industriels.

IEC/CEI 62443 : Le document considéré comme le plus emblématique à ce jour est probablement la série de normes ISA99 [ISA99, 2007], débutée en 2007. Ces documents ont par la suite été refondus en la série de norme IEC/CEI 62443 [IEC-62443, 2010] en 2010 et représentent un volume de plus de 1000 pages. Ces normes, à portée internationale, sont transverses à tous les types d'industries. Elles se destinent aussi bien aux organisations qui possèdent des systèmes industriels qu'aux intégrateurs qui les installent. Le consortium pilotant la rédaction de ces documents est essentiellement constitué d'industriels et comprend accessoirement quelques acteurs gouvernementaux parmi les auteurs.

ISO 27019 : Ce document publié en 2013 [ISO-27019, 2013] fait partie de la série des ISO 27000 [ISO-27000, 2005], qui traitent de la sécurité des systèmes d'information. Il propose notamment d'étendre aux systèmes industriels les idées des documents précédents. Il est clairement destiné aux Responsables de la Sécurité des Systèmes d'Information (RSSI), déjà familiers de cette famille de normes, en reprenant notamment la structure de l'ISO 27002 [ISO-27002, 2005] qui fait référence pour les systèmes d'information classiques. Cette norme, à portée internationale, est transverse à tous les types d'industrie. Le consortium pilotant la rédaction de ce document est essentiellement constitué d'industriels et comprend accessoirement quelques acteurs gouvernementaux parmi les auteurs.

ENISA– Protecting Industrial Control Systems : Entre 2011 et 2013, l'Agence Européenne chargée de la sécurité des réseaux et de l'information (ENISA) publie une série

2. http://securid.novaclic.com/securite-internet/referentiels-securite-si.html
3. https://clusif.fr/content/uploads/2015/09/clusif-SCADA-2014-GBI-SOL.pdf
4. https://scadahacker.com/library/

de guides sur la sécurité des systèmes industriels. Le premier document [Leszczyna et al., 2011] est avant tout stratégique et s'adresse aux décideurs politiques. Les documents suivants [Cimpean et al., 2012, Pauna and Moulinos, 2013] proposent des mesures plus détaillées et centrées sur une problématique particulière (smart-grid, patch-management). La rédaction de ces guides, à portée européenne, est pilotée par une agence publique et inclut des auteurs industriels.

IEC/CEI 62541 : Publiée entre 2015 et 2016, la norme internationale IEC/CEI 62541 [IEC-62541, 2015] est la refonte des spécifications techniques du protocole de communication industriel OPC-UA. Les premières publications de ce document ont été rendues publiques entre 2010 et 2011 par la Fondation OPC [5] (un consortium industriel). Dû au fait que le protocole OPC-UA est extrêmement polyvalent, cette norme aborde au final des sujets variés tels que la gestion des identifiants ou la cryptographie. Par ailleurs, ce protocole étant parmi les premiers à proposer ce type de fonctionnalité de sécurité dans le cadre des protocoles industriels, ces documents abordent des problématiques très récentes pour les systèmes industriels. Le consortium pilotant la rédaction de ce document est essentiellement constitué d'industriels et comprend accessoirement quelques acteurs gouvernementaux parmi les auteurs.

Les normes et guides décrits ci-dessus sont internationaux mais de nombreux pays ont proposé des documents similaires via leurs agences gouvernementales respectives. Ces documents nationaux ont avant tout pour but de sensibiliser les acteurs industriels de leur pays afin de limiter la portée d'éventuelles attaques. De façon intéressante, ils sont aussi souvent traduits pour être disséminés à l'international, implicitement pour montrer la capacité du pays à se défendre.

ANSSI– Maîtriser la SSI pour les systèmes industriels : En France, l'Agence Nationale de la Sécurité des Systèmes d'Information (ANSSI) a publié une série de quatre guides dédiés aux systèmes industriels [ANSSI, 2012a, ANSSI, 2012b, ANSSI, 2014a, ANSSI, 2014b] entre 2012 et 2014 qui ont ensuite été traduits en anglais. Ces documents ont un contenu très varié allant d'une introduction grand public à un ensemble de mesures détaillées à appliquer aussi bien par les entités responsables de systèmes industriels que par les intégrateurs et les évaluateurs. On y trouve aussi un cas d'exemple pratique appliquant concrètement le processus d'évaluation et les contre-mesures à un système industriel. La rédaction de ces guides est pilotée par une agence gouvernementale et inclut des auteurs industriels.

NIST– SP800-82 : Aux États-Unis, le *National Institute of Standards and Technology* (NIST) a publié en 2011, puis revu en 2013 et 2015, un guide sur la sécurité des systèmes industriels [Stouffer et al., 2011]. Son contenu est très similaire aux normes IEC/CEI 62443 et ISO 27019. Il est cependant beaucoup plus succinct (155 pages) et est souvent considéré comme un très bon document d'introduction pour le public anglophone. De la même façon que l'ISO 27019, ce guide est construit sur le modèle de son pendant

5. https://opcfoundation.org/

destiné aux systèmes d'information (SP800-53 [Ross et al., 2013]). La rédaction de ce document est pilotée par une agence gouvernementale et inclut des auteurs industriels.

CPNI– Process Control and SCADA Security : Entre 2008 et 2011, le *Center for the Protection of National Infrastructure* (CPNI) au Royaume-Uni a publié une série de 8 guides sur la sécurité des systèmes industriels [CPNI, 2008]. Ils offrent une vision très haut niveau en proposant des bonnes pratiques stratégiques, managériales et organisationnelles. Les thèmes abordés vont de l'analyse de risques à la gestion des risques liés aux tierces parties en passant par la réponse à incident. Ces guides sont complétés par des documents plus récents, qui donnent les informations techniques sur certaines problématiques telles que les audits de sécurité des systèmes industriels en 2011 [CPNI, 2011]. La rédaction de ce document est pilotée par une agence gouvernementale et inclut des auteurs industriels.

BSI– CIP Implementation Plan of the National Plan for Information Infrastructure Protection : En 2009, le *Bundesministerium des Innern* (BMI) et le *Bundesamt für Sicherheit in der Informationstechnik* (BSI) publient en coopération un guide sur la sécurité des systèmes industriels [BSI, 2009]. Similaire aux guides du CPNI, ce document reste très haut niveau et ne rentre que très peu dans les aspects techniques. La rédaction de ce document est pilotée par une agence gouvernementale et inclut des auteurs industriels.

Enfin, les documents décrits ci-dessus sont essentiellement transverses à tous les domaines d'industrie. Cependant, chaque domaine (nucléaire, ferroviaire, énergétique, etc.) a des particularités qui nécessitent des protections adaptées, relativement à celles décrites dans les documents transverses. Nous en décrivons quelques normes métier à titre d'exemple.

Nucléaire : La norme internationale IEC/CEI 62645 [IEC-62645, 2014] a été publiée entre 2008 et 2014 et est la référence en matière de sécurité des systèmes industriels dans le domaine du nucléaire. Elle propose des recommandations similaires à celles de la norme IEC/CEI 62443 telles que le cloisonnement en zones de différente sécurité. Elle met cependant un accent particulier sur le fait que les mesures de sécurité ne doivent pas venir à l'encontre des mesures de sûreté de fonctionnement. Le consortium pilotant la rédaction est essentiellement constitué d'industriels et comprend accessoirement quelques acteurs gouvernementaux parmi les auteurs. Dans un second temps, le même consortium entame la rédaction de la norme IEC/CEI 62859 [IEC-62859, 2016], basée sur la norme IEC/CEI 62645. Avec une date de stabilité affichée pour 2020, cette norme se concentre sur les relations entre sûreté de fonctionnement et sécurité.

Domaine énergétique : La norme internationale IEC/CEI 62351 [IEC-62351, 2016] est publiée entre 2007 et 2016 et vise à développer la sécurité des protocoles des communications utilisés dans la distribution d'électricité (*power grid*). Visant des protocoles du domaine tels que MMS, GOOSE ou encore DNP3, elle propose des mesures de protection en accord avec l'état de l'art en matière de sécurité (utilisation du protocole TLS, infrastructure à clés publiques, etc.). Le consortium pilotant la rédaction

est essentiellement constitué d'industriels et comprend accessoirement quelques acteurs gouvernementaux parmi les auteurs. Un autre document pouvant être référencé est le guide NISTIR 7628 [Lee and Brewer, 2010] issu du NIST américain.

Ferroviaire : Au Royaume-Uni, le *Rail Safety and Standards Board* (RSSB) et le *Department for Transport* publient en 2016 un guide nommé *Rail Cyber Security Guidance to Industry* [RSSB, 2016]. Similaire aux guides du CPNI, ce document reste très haut niveau et ne rentre que très peu dans les aspects techniques. La rédaction de ce document est pilotée par une agence gouvernementale et inclut des auteurs industriels.

Sécurité aérienne : Aux États-Unis, le *Cyber Security Forum Initiative* (CSFI) publie en 2015 un guide intitulé *CSFI ATC (Air Traffic Control) Cyber Security Project* [CSFI, 2015]. Ce document propose une analyse en profondeur et très technique des risques et des contre-mesures ciblant le domaine du contrôle aérien. Il présente notamment un intérêt du fait que les communications de ce domaine sont essentiellement sans fil, ce qui tranche des autres industries. Le consortium pilotant la rédaction est essentiellement constitué d'industriels et comprend accessoirement quelques acteurs gouvernementaux parmi les auteurs.

Pétrole et gaz : Toujours aux États-Unis, l'*American Gas Association* (AGA) publie en 2006 une série de guides nommée *Cryptographic Protection of SCADA Communications* [AGA, 2006]. Ces guides y analysent notamment la possibilité d'ajouter des *Hardware Security Modules* (HSM) aux automates programmables et SCADA déjà existants pour leur permettre d'utiliser des primitives cryptographiques. Le consortium pilotant la rédaction est essentiellement constitué d'industriels et comprend accessoirement quelques acteurs gouvernementaux parmi les auteurs.

La multitude de documents (normes et guides) existant montre l'importance du domaine et de ses enjeux. On peut également noter une bonne répartition entre les acteurs gouvernementaux et industriels lors de la rédaction de ces documents, témoignant que cette prise de conscience est collective à de multiples strates de la société. Cependant, comme le montrent notamment Piètre-Cambacédès et Chaudet ainsi que le CLUSIF, le nombre important de documents peut également être contre-productif car ils n'utilisent pas toujours le même vocabulaire et peuvent parfois donner des conseils contradictoires. Enfin, ces documents sont généralement écrits de façon à rester très haut niveau, afin de laisser libre l'application des recommandations suivant le contexte. Ainsi, ils ne donnent généralement que peu de solutions concrètes et doivent donc être utilisés comme base de travail pour des mécanismes applicables.

1.3. Une variété d'attaquants

Le terme « *hacker* », souvent associé à une personne vêtue d'une cagoule ou d'une capuche et attaquant des systèmes depuis une pièce sombre, a profondément divergé de son sens initial. Ce terme vient à l'origine du verbe *to hack*, signifiant « découper » en vieil anglais. En sécurité informatique, il s'appliquait ainsi à des personnes cherchant à

comprendre le fonctionnement profond des systèmes et des réseaux et de résoudre les problèmes par des moyens non conventionnels. Il impliquait notamment de se conformer à une certaine éthique, par exemple décrite dans [Blankenship, 1986]. L'informatique étant de plus en plus présent dans nos vies, d'autres profils d'attaquants ont fait leur apparition. Par ailleurs, de nombreuses vulnérabilités et exploits peuvent facilement être trouvées sur Internet. Il existe également des entreprises qui achètent et revendent ces vulnérabilités au plus offrant. Ainsi, on doit maintenant faire face à une multitude d'attaquants avec des objectifs, des capacités et des moyens très variables, allant des *script-kiddies* aux gouvernements en passant par les hacktivistes, les mafias, ou encore les organisations terroristes [Holt and Kilger, 2012]. Les catégories de « hackers » sont par ailleurs souvent distinguées par des couleurs de chapeau, à la manière des films de western. On se propose ici de décrire succinctement ces attaquants afin d'illustrer leur variété.

Consultants en sécurité : Aussi appelés *white hats*, ce sont des professionnels qui testent la sécurité des systèmes afin de faire remonter les vulnérabilités aux éditeurs. Il peuvent soit appartenir à une entreprise spécialisée, soit travailler de façon indépendante (par exemple via des programmes de *bug bounty* (chasse de primes aux bugs)). De plus en plus d'entreprises organisent des campagnes de tests *Red/Blue teams* faisant s'opposer une équipe d'attaquants et une équipe de défenseurs afin de tester la réactivité de leurs défenses.

Autres consultants : Également consultants mais pas nécessairement en sécurité informatique, ils sont chargés de trouver des bogues pouvant être liés à de la sécurité. Microsoft utilise notamment le terme *blue hats* pour désigner ces équipes.

Les deux types d'attaquants décrits ci-dessus agissent uniquement dans le cadre de la loi afin d'aider à sécuriser les systèmes. Il arrive cependant que certains attaquants outrepassent par moment la loi sans nécessairement avoir d'objectifs malicieux.

***Grey hats* :** Contrairement aux consultants qui attaquent les systèmes à la demande de leurs propriétaires, les *grey hats* agissent le plus souvent par eux-mêmes. Ils font fréquemment remonter les failles qu'ils trouvent aux éditeurs et profitent parfois de l'occasion pour se faire connaître (en laissant par exemple des traces dans le système).

Hacktivistes : Ce terme est un mot-valise, contraction de hacker et activistes. Les *hacktivistes* sont principalement des *grey hats* qui agissent pour la promotion d'une cause (politique, sociétale, environnementale, religieuse, etc.).

Le reste des catégories ci-dessous peuvent être regroupées sous l'appellation *black hats*, qui agissent essentiellement hors-la-loi.

***Script-kiddies* :** Aussi appelés *skids*, *skiddies* ou *lamers*, ce sont des attaquants avec peu ou pas de compétences qui utilisent des outils automatiques permettant d'attaquer les systèmes. Ils agissent essentiellement pour se faire connaître et sont souvent dénigrés par la communauté hacker pour leur manque de maturité.

Employés mécontents : Des employés mécontents peuvent vouloir se venger de leur employeur et profiter pour cela de leurs accès aux systèmes. C'est par exemple le cas de l'auteur de l'attaque Maroochy Shire, décrite en section 1.4, et qui servira de fil rouge aux contributions de cette thèse. Ces attaquants sont souvent considérés comme potentiellement très dangereux de part leur position à l'intérieur du système et ce, peu importe leur niveau de compétence.

Mafias : Organisations criminelles agissant essentiellement par appât du gain, les mafias ont rapidement compris qu'il était possible de transposer leurs activités dans le monde informatique. Elles peuvent par exemple être la source de campagnes de rançongiciel (*ransomware*) qui prennent en otage les données des victimes et proposent de les rendre contre de l'argent.

Organisation terroristes : Similairement à la différence entre les *grey hats* et les hacktivistes, les organisations terroristes se distinguent des mafias par leur objectif de promouvoir une cause par la violence. Contrairement aux hacktivistes, ils n'hésitent pas à briser la loi dans le but de détruire les systèmes des victimes. On peut par exemple citer l'attaque contre TV5 Monde, revendiquée par des groupes se réclamant de l'organisation État islamique (bien qu'il ne soit pas certains qu'ils en soient réellement la cause). D'autres attaques de ce genre ont été menées afin de tenter de prendre le contrôle de comptes de quotidiens d'informations sur les réseaux sociaux.

Il existe enfin un dernier type d'attaquant plus difficile à classer dans les catégories *white/grey/black hats*, les acteurs étatiques.

Acteurs étatiques : Agissant généralement dans le secret, il est difficile d'établir l'ancienneté de ces attaquants et de leur attribuer des attaques ayant eu lieu. Il est compliqué de classer ces attaquants comme bons ou mauvais puisqu'ils agissent généralement à la fois sous le contrôle d'un état et contre la loi d'un autre. Ces attaquants sont par contre unanimement reconnus comme extrêmement puissants de part leur niveau de compétence très élevé et les moyens dont ils peuvent disposer. Plusieurs états sont par ailleurs accusés de soutenir des groupes d'attaquants afin de ne pas être directement impliqués dans les attaques menées. L'attaque Stuxnet ayant visé le programme nucléaire iranien serait imputée à des agences gouvernementales américaines et israéliennes, bien que cela n'ait jamais été ni prouvé ni reconnu.

1.4. Sélection d'exemples d'attaques

Cette section présente une liste non exhaustive d'attaques ou de pannes ayant touché des systèmes industriels par le passé, ceci afin de donner une meilleure idée au lecteur de leurs causes et conséquences. Certains exemples présentés ici ont été choisis parce qu'ils présentent un intérêt pour motiver les contributions de cette thèse. D'autres servent simplement à montrer les différentes formes qu'une attaque peut prendre. Une liste plus complète d'exemples peut être trouvée dans [Weiss, 2010] et [Hayden et al., 2014].

Attaque 1 – Maroochy Shire, 2000 : Un ancien employé de Hunter Watertech (maintenant HWT), sous-traitant du conseil du Comté de Maroochy Shire (maintenant Sunshine Coast) déverse le contenu d'une cuve d'eaux usagées dans la nature. Il utilise pour cela du matériel provenant de HWT à savoir un RTU PDS Compact 500 (le même que ceux contrôlant les cuves), un émetteur-récepteur radio (nécessaire aux communications sans-fil) et un ordinateur portable avec un logiciel de HWT servant à reprogrammer les RTU [Weiss, 2010]. Le RTU PDS Compact 500 de HWT communique via les protocoles MODBUS et DNP3 [Clarke et al., 2004]. Durant l'attaque qui s'est déroulée sur plusieurs mois, le système d'épuration ciblé a connu une série de défauts incluant :

— des pompes qui ne fonctionnaient pas lorsqu'elles auraient dû ;

— des alarmes qui n'étaient pas remontées au SCADA ;

— le SCADA qui n'était pas en mesure de communiquer avec des pompes.

A la suite de ces défauts (aujourd'hui connus comme étant les conséquences de l'attaque), une cuve a débordé. D'après les conclusions d'un représentant de l'Agence Australienne de Protection de l'Environnement (EPA), l'eau de la crique est devenue noire, causant plusieurs inondations dans des sous-sols et provoquant la mort de la faune marine. A notre connaissance, aucune analyse de l'attaque [Slay and Miller, 2007, Abrams and Weiss, 2008, Weiss, 2010] ne décrit précisément quel enchaînement d'actions a conduit au débordement d'une cuve. Cependant, au vu des défauts qu'a connu le système, on peut *supposer* qu'une pompe était sensée vider la cuve à mesure qu'elle se remplissait d'eaux usagées. Le fait qu'elle ne démarre pas aurait fait déborder la cuve. Par ailleurs, dû au fait que les alarmes n'étaient pas remontées au SCADA, les opérateurs ne se seraient aperçus des problèmes qu'une fois les dégâts causés. Une analyse de l'attaque sous hypothèse de cette suite d'actions est proposée en annexe A.3.

L'attaque était intentionnelle, ciblée, avec un objectif et commise par un ancien employé pouvant donc être considéré comme une menace interne. L'attaquant connaissait le système ciblé, ainsi que ses vulnérabilités. Les protocoles utilisés par le système (MODBUS ou DNP3) ne prévoyaient aucun mécanisme de sécurité, rendant possible l'envoi de commandes arbitraires. Manifestement aucun mécanisme de protection n'a empêché la cuve de se remplir alors que les pompes étaient à l'arrêt à l'exception des alarmes qui n'ont pas pu être remontées au SCADA. Enfin l'attaque s'étant étalée sur une période de plusieurs mois montre l'absence d'entraînement du personnel et d'analyses de risques en amont [Weiss, 2010]. Ainsi, dans le cadre de nos travaux, nous pouvons noter les vulnérabilités et contre-mesures possibles suivantes :

Vulnérabilités :

— Absence de mécanisme de sûreté empêchant de répandre le contenu de la cuve.

— Absence d'authentification dans les protocoles de communication.

— Absence de préparation des employés à une attaque.

Contre-mesures :

— Filtrer les communications à l'aide de règles métier pour interdire des commandes pouvant amener à déverser la cuve.

— Utiliser des protocoles sécurisés et vérifiés à l'aide d'outils du domaine.
— Étudier les scénarios d'attaques possibles afin de pouvoir protéger le système en amont ou à minima entraîner le personnel à réagir lorsque l'un d'eux se produit.

Attaque 2 – Jeep Cherokee, 2013, 2015 : Charlie Miller et Chris Valasek, aujourd'hui chercheurs en sécurité chez Uber, montrent à deux occasions comment prendre le contrôle total de voitures connectées [Greenberg, 2013, Greenberg, 2015]. Les deux démonstrations sont couvertes par le journaliste Andy Greenberg au volant des voitures. La première démonstration en 2013 se fait sur un véhicule *Ford Escape* grâce à un ordinateur relié directement aux systèmes de contrôle de la voiture. La seconde démonstration en 2015 se fait, quant à elle, sur une *Jeep Cherokee* via Internet en exploitant une vulnérabilité *zero-day*. Dans les deux cas, ils sont en mesure d'injecter des paquets CAN bus contenant des commandes malicieuses. Ils ont ainsi pu allumer la radio de la voiture, mettre hors-service les freins ou l'accélérateur, contrôler le volant, etc. Toujours en 2015, Kevin Mahaffey et Marc Rogers présentent une étude similaire sur un véhicule *Tesla Model S* [Mahaffey, 2015].

Les composants électroniques des véhicules (appelés *Electronic Control Units* – ECU) communiquent entre eux via le protocole CAN bus. Ce protocole publié en 1986 ne prévoit aucun mécanisme de sécurité, rendant possible l'envoi de commandes arbitraires. Il est difficile d'argumenter sur l'absence de mécanismes de sûreté de part la complexité du procédé que représente la voiture et les quantités de tests qu'elles doivent passer pour être mises en circulation. Cependant il parait tout de même troublant qu'il soit possible de couper le moteur ou de désactiver les freins alors que la voiture roule à pleine vitesse, comme dans le cas des démonstrations sur les véhicules *Ford Escape* et *Jeep Cherokee*. Enfin, dans les trois démonstrations, des commandes malicieuses ont pu être envoyées soit en compromettant un composant soit en se connectant directement au bus de contrôle. Cela témoigne d'une absence d'étude des conséquences de la compromission ou de l'insertion d'un composant. Ainsi, dans le cadre de nos travaux, nous pouvons noter les vulnérabilités et contre-mesures possibles suivantes :

Vulnérabilités :

— Absence de mécanisme de sûreté empêchant à minima certains comportements dangereux du procédé (désactivation des freins, etc.).
— Absence d'authentification dans les protocoles de communication.
— Absence d'étude des conséquences de la compromission ou de l'insertion d'un composant.

Contre-mesures :

— Filtrer les communications à l'aide de règles métier pour interdire à minima certaines commandes dangereuses (désactivation des freins, etc.).
— Utiliser des protocoles sécurisés et vérifiés à l'aide d'outils du domaine.
— Étudier les scénarios d'attaques possibles afin de pouvoir les corriger en amont.

Attaque 3 – Aurora, 2007 : En 2007, le projet Aurora [Aurora Project, 2014, Weiss, 2010], mené par le Laboratoire national de l'Idaho (projet classé secret défense puis déclassifié en 2014) a démontré la nécessité de protéger les équipements industriels contre

des attaques informatiques. Ce test a été mené sur un générateur diesel contrôlé par des disjoncteurs. Des chercheurs ont envoyé des commandes d'ouverture et de fermeture aux disjoncteurs de façon désynchronisée. Ces ordres ont eu des conséquences physiques sur les mécanismes de rotation du générateur, provoquant sa destruction. Cette démonstration a été réalisée par des opérateurs légitimement connectés au système (pouvant être vus comme des opérateurs malicieux). Cependant, les disjoncteurs du générateur sont contrôlés via le protocole MODBUS qui ne prévoit aucun mécanisme de sécurité. Ainsi, la même attaque aurait pu être réalisée par un attaquant présent sur le réseau et pouvant envoyer des commandes arbitraires (en particulier dans ce cas, à des moments arbitraires). Ce type d'attaque peut être évité en rejetant les flux qui ne garantissent pas un certain délai entre les commandes d'ouverture et de fermeture (permettant la synchronisation nécessaire au mécanismes de rotation). Ainsi, dans le cadre de nos travaux, nous pouvons noter les vulnérabilités et contre-mesures possibles suivantes :

Vulnérabilités :

— Absence de mécanisme de sûreté empêchant l'envoi de commandes désynchronisées.

— Absence d'authentification dans les protocoles de communication.

Contre-mesures :

— Filtrer les communications à l'aide de règles métier pour interdire les commandes qui seraient désynchronisées.

— Utiliser des protocoles sécurisés et vérifiés à l'aide d'outils du domaine.

Attaque 4 – Stuxnet, 2005-2010 : Entre 2005 et 2010, l'Iran a été victime d'une attaque contre son programme nucléaire. Beaucoup d'informations circulent sur Stuxnet [Falliere et al., 2011, Langner, 2011] et, bien que la majorité des experts en sécurité convergent, très peu de ces informations ont été officiellement reconnues. En particulier l'origine de l'attaque serait attribuée aux gouvernements américain et israélien, bien que ni l'un ni l'autre ne le reconnaissent officiellement. Si cela est vrai, Stuxnet serait la première attaque étatique connue. D'après le rapport technique de Symantec [Falliere et al., 2011], le vers a pu être introduit via une clé USB infectée appartenant à un sous-traitant. Les binaires malveillants étaient signés par des certificats compromis de *Realtek Semiconductor Corps* pour ne pas éveiller les soupçons. Une fois dans le système, le vers s'est propagé notamment via le réseau et les médias amovibles pour atteindre un ordinateur servant à programmer des automates Siemens Simatic S7-315. Une fois l'une de ces machines atteintes, le vers l'a utilisée pour modifier le programme exécuté par l'automate programmable. Cela lui a permis notamment d'intercepter les messages envoyés et reçus par l'automate et de les remplacer.

Avant toute action, le virus a surveillé le trafic afin de déterminer si la cible était dans un certain état visé. Toujours d'après [Falliere et al., 2011], cette phase d'observation a pu durer de treize jours à trois mois. Le virus est ensuite passé en phase d'attaque en modifiant les valeurs des variables contrôlant la vitesse de rotation des centrifugeuses. Ainsi, Stuxnet a saboté le système en ralentissant et en accélérant les centrifugeuses à différents moments. L'attaque menée via Stuxnet est extrêmement complexe à de multiples points de vue :

— préparation minutieuse de l'attaque ;

— structure du virus ;
— méthode d'infection (quatre vulnérabilités zero-day utilisées) ;
— méthode de reproduction ;
— capacité à masquer ses actions ;
— infection à grande échelle mais exécution uniquement sur les machines de l'installation visée.
— etc.

Cette complexité permet d'analyser l'attaque dans une multitude de contextes (*reverse engineering*, réseau, *social engineering*, génie nucléaire, etc.). Cependant il nous semble, d'une part, qu'aucune de ces analyses n'est vraiment pertinente seule et que, d'autre part, la diversité de ces contextes rend très difficile toute analyse globale. De plus, de part les moyens humains et financiers mis en oeuvre, il est peut probable que les contre-mesures résultant de ces analyses auraient pu suffire à éviter l'attaque. Aussi toute analyse de cette attaque, bien que très symbolique pour la communauté, doit être prise avec précaution (y compris pour les vulnérabilités et contre-mesures ci-après). Ainsi, dans le cadre de nos travaux, nous pouvons noter les vulnérabilités et contre-mesures possibles suivantes :

Vulnérabilités :

— La propagation du virus par le réseau et des médias amovibles.
— La capacité du virus à modifier les vitesses de rotation par l'intermédiaire des commandes envoyées.

Contre-mesures :

— Cloisonner l'installation en zones et vérification des informations transmises d'une zone à une autre pour éviter la propagation du virus.
— Filtrer les communications à l'aide de règles métier pour interdire les commandes modifiant la vitesse des centrifugeuses.

Une bonne pratique pour limiter l'impact de la première vulnérabilité est d'opérer un cloisonnement entre des zones de sécurité différente, à fortiori la partie opérationnelle de l'usine et les zones ayant accès à internet. Concernant la seconde vulnérabilité : une fois le virus installé sur les automates programmables, il communique directement avec d'autres automates (c'est à dire des communications internes au niveau 1). Ces communications sont très difficiles à filtrer de part la multitude de protocoles propriétaires utilisés pour chaque modèle d'automate et des temps de latence imposés [6].

Attaque 5 – Brown Ferry, 2006 : Le 19 août 2006, l'unité 3 de la centrale nucléaire de Brown Ferry a subi un arrêt manuel d'urgence (SCRAM) [NRC, 2007, Weiss, 2010, Hardy, 2012]. Cette procédure a fait suite à une panne simultanée de deux pompes de refroidissement du réacteur. Après investigation, il s'est avéré que ce sont les variateurs électroniques de vitesse (VFD) contrôlant les pompes qui ont cessé de répondre. Un automate programmable contrôlant l'adoucissement de l'eau est par ailleurs tombé en panne en même temps que les deux VFD. La cause de la panne des VFD était une

6. Elles sont notamment hors de portée du dispositif de filtrage présenté en chapitre 3.

saturation du réseau – *broadcast storm*. Les techniciens de la centrale n'ont pas été en mesure de déterminer si la saturation a été causée par l'automate en panne ou s'il en a également été victime. Le problème a été corrigé via un mécanisme pour limiter le trafic sur le réseau et en plaçant les VFD et l'automate chacun derrière un pare-feu. Cet événement est une panne est non une attaque. Cependant, elle aurait parfaitement pu être effectuée par un intrus ayant réussi à entrer sur le réseau (par exemple en corrompant l'automate à la manière de Stuxnet). Ainsi, dans le cadre de nos travaux, nous pouvons noter les vulnérabilités et contre-mesures possibles suivantes :

Vulnérabilités :

— Interconnexion de composants vitaux de la centrale (les VFD) avec d'autres moins importants.

— Absence de mécanisme pour contrôler le trafic.

Contre-mesures :

— Cloisonner l'installation en zones (il est probable que le pare-feu ne résiste pas à la saturation du réseau, mais il n'est pas un composant vital à la centrale).

Attaque 6 – San Diego, 1999 : En novembre 1999, la régie des eaux de San Diego (*San Diego County Water Authority*) a connu d'importantes perturbations de son réseau sans-fil [Foster Jr et al., 2008, Weiss, 2010]. Les mêmes perturbations ont été subies par la compagnie de gaz et électricité (*San Diego Gas and Electric*). Dans les deux cas, les opérateurs ont perdu la capacité d'envoyer des commandes d'ouverture et de fermeture à des valves critiques et ont dû dépêcher des techniciens pour agir manuellement. Après enquête, la source des perturbations s'est avérée être un radar équipé sur un navire de la marine américaine, patrouillant à 40 kilomètres des côtes. A nouveau, la panne n'est pas due à un acte malveillant mais un attaquant en possession d'un matériel suffisamment puissant aurait pu générer les mêmes perturbations.

Vulnérabilités :

— Utilisation de médium de communications sans-fils sensibles à des perturbations électromagnétiques.

Contre-mesures :

— Utiliser des médiums de communication résistants tels que des fibres optiques.

La table 1.1 résume les différents exemples décrits ci-dessus. Leur description et leur analyse dans cette section restent les plus factuelles possibles afin d'être utilisés dans la section 2.3 pour motiver les objectifs de la thèse. De façon intéressante, dans [Weiss, 2010], Weiss présente une liste d'incidents non intentionnels et d'attaques contre des systèmes industriels. Il conclut en expliquant que les attaques sont plutôt rares, ces derniers étant beaucoup plus impactés par les incidents non intentionnels. Cependant, en accord avec la liste établie par le SANS (*SysAdmin, Audit, Network and Security*) [Hayden et al., 2014], au moins dix-sept attaques auraient été perpétrées contre des systèmes industriels entre Maroochy Shire en 2000 et la publication du livre de Weiss en 2010. L'affirmation de Weiss, que les attaques sont rares, peut néanmoins se justifier du fait

que leur impact est essentiellement resté limité à des dénis de service ou à de l'espionnage industriel, portant surtout atteinte à l'image de marque. Stuxnet est effectivement reconnu comme étant la première attaque médiatisée avec pour but le sabotage.

Quoi qu'il en soit, la tendance est clairement en train de s'inverser et les attaques contre des systèmes industriels deviennent de plus en plus fréquentes. Entre 2011 et 2016, les virus Duqu, Flame, Havex et Black Energy ont été utilisés dans des campagnes de hameçonnage ciblées sur des employés de systèmes industriels. Une fois téléchargés, ces virus permettent de faire de la reconnaissance et donnent le plus souvent un accès distant aux attaquants. Une fois dans le systèmes, les attaquants peuvent alors planifier une attaque puis la lancer. Ce fut par exemple le cas en 2014 sur une aciérie allemande [Lee et al., 2014] et en 2016 en causant un black-out en Ukraine [Lee et al., 2016]. A notre connaissance, Stuxnet reste cependant la seule attaque pilotée automatiquement par le virus lui-même.

ID	Nom	Type d'attaque	Vulnérabilités	Contre-mesures possibles
1	Maroochy Shire	Envoi de commandes malicieuses	• Manque de sûreté. • Protocoles non sécurisés. • Manque de préparation.	• Filtrage métier. • Protocoles sécurisés et vérifiés. • Étude des scénarios d'attaques.
2	Jeep Cherokee	Envoi de commandes malicieuses	• Manque de sûreté. • Protocoles non sécurisés. • Manque de préparation.	• Filtrage métier. • Protocoles sécurisés et vérifiés. • Étude des scénarios d'attaques.
3	Aurora	Envoi de commandes désynchronisées	• Manque de sûreté. • Protocoles non sécurisés.	• Filtrage métier. • Protocoles sécurisés et vérifiés.
4	Stuxnet	Chargement d'un virus	• Propagation par le réseau. • Défaut de sûreté.	• Cloisonnement de l'installation. • Filtrage métier.
5	Brown Ferry	Saturation du réseau	• Interconnexion de composants. • Absence limitation de trafic.	• Cloisonnement de l'installation.
6	San Diego	Interférences électromagnétiques	• Perturbations électromagnétiques.	• Médiums insensibles (fibre optique).

TABLE 1.1. – Récapitulatif des attaques

Chapitre 2

Spécificités des systèmes industriels et objectifs de la thèse

> Do we realize that industry, which has been our good servant, might make a poor master ?
>
> *(Aldo Leopold, 1925)*

Résumé du chapitre

Les systèmes industriels ont des spécificités qui seront référencées tout au long de cette thèse. Ce chapitre introduit donc ces spécificités, à savoir, le fonctionnement des automates programmables et le format des messages échangés. De plus, nous motivons les contributions de cette thèse en se basant sur des attaques ayant eu lieu.

Sommaire

2.1. Fonctionnement des automates programmables

Les automates programmables sont des équipements industriels opérant à la couche 1 (Traitement d'automatisme) de la classification de Purdue présentée en figure 1.1. Comme en section 1.1.1, ils font le lien entre les équipements de supervision (SCADA) du niveau 2 et les capteurs et actionneurs du niveau 0. Ils doivent donc interpréter des ordres généraux du SCADA tels que « démarrer la pompe » et déterminer quels actionneurs impacter pour exécuter l'ordre. Pour cela, ils exécutent un programme à la manière des ordinateurs.

2.1.1. Composition d'un automate programmable

La figure 2.1 décrit les composants internes d'un automate programmable. Á la manière d'un ordinateur, un automate est composé d'une unité centrale à laquelle pourront être connectés des modules ayant des fonctionnalités particulières. L'unité centrale est généralement constituée d'un processeur (CPU), de mémoires et d'un module permettant de reprogrammer l'automate. Le processeur est chargé d'exécuter le programme qui est stocké sur une mémoire morte reprogrammable (EPROM). Une mémoire morte (ROM) contient le système d'exploitation de l'automate tandis qu'une mémoire vive (RAM) contient, entre autres, les variables utilisées dans le programme. Ces variables représentent généralement les valeurs des capteurs et des actionneurs connectés à l'automate mais peuvent aussi représenter des objets plus complexes contrôlant plusieurs capteurs et actionneurs à la fois. Un module de reprogrammation permet de reprogrammer l'automate (c'est-à-dire de modifier le programme qu'il exécute), le plus souvent en le connectant à une station dédiée. Le paradigme dans lequel les programmes sont décrits varie selon les constructeurs et selon les zones géographiques. On peut par exemple citer le format *Grafcet* (dérivé des réseaux de Petri), le texte structuré (qui se présente comme une langage de programmation) ou encore la logique en échelle – *ladder logic* – (qui ressemble aux schémas de circuits électriques). Ces paradigmes sont synthétisés dans le standard IEC/CEI 61131-3 [IEC-61131-3, 2013].

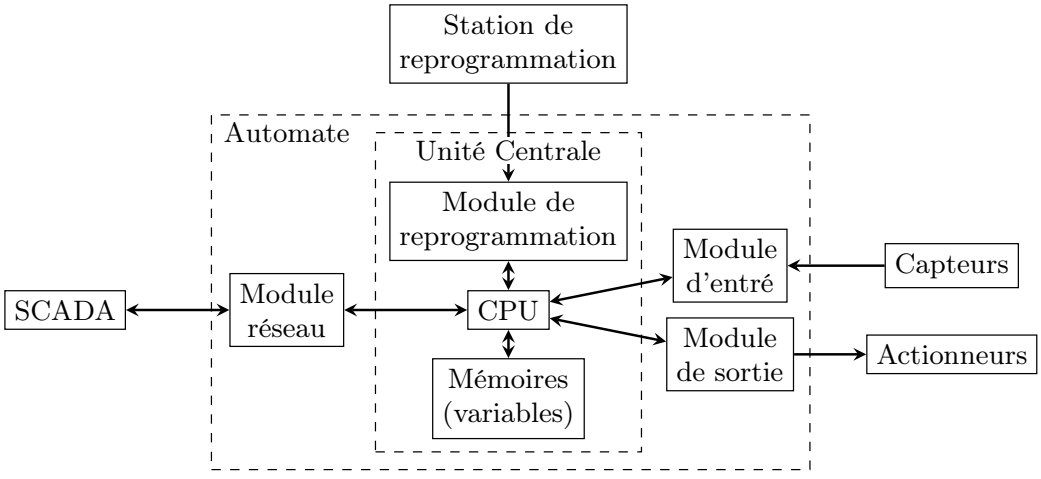


Figure 2.1. – Composants d'un automate programmable

Des modules sont ensuite ajoutés à la manière des cartes PCI d'un ordinateur. En particulier, les connexions entre les automates et les capteurs et actionneurs sont souvent des courants électriques. Un module d'entrée permettra de convertir les courants émis par les capteurs en valeurs pour les variables de l'automate. De façon analogue, un module de sortie permettra d'émettre des signaux électriques afin de modifier l'état des actionneurs. Enfin, un module réseau permettra de communiquer avec l'extérieur (généralement avec un SCADA ou avec d'autres automates). C'est ce module qui réceptionne les ordres envoyés par le SCADA et les transfère au processeur comme des entrées du programme. Les variables manipulées par l'automate sont organisées dans un *espace d'adressage* dont la structure dépend de l'automate et du protocole de communication qu'il utilise pour communiquer. Cette notion d'espaces d'adressage sera expliquée plus en détail pour les protocoles MODBUS (en section 2.2.3.1) et OPC-UA (en section 2.2.3.2). Ainsi, les ordres donnés par les SCADA sont sous la forme de lecture et d'écriture des variables de l'automate. L'ordre « démarrer la pompe » pourrait par exemple être traduit par l'écriture du booléen *true* sur une variable dédiée. De façon analogue, le SCADA fera périodiquement des lectures de certaines variables afin d'agréger et de présenter l'état global du système aux opérateurs. Ces requêtes de lecture et d'écriture seront instanciées en section 2.1.2 décrites de façon plus détaillée en section 2.2.2.

2.1.2. Exemple de l'attaque Maroochy Shire

Pour illustrer le fonctionnement d'un automate programmable et les communications qu'il effectue avec le SCADA, les capteurs et les actionneurs, on se propose de montrer l'exemple du système victime de l'attaque Maroochy Shire. La figure 2.2 montre une vue synoptique du système, telle qu'elle pourrait être visualisée sur un écran du SCADA.

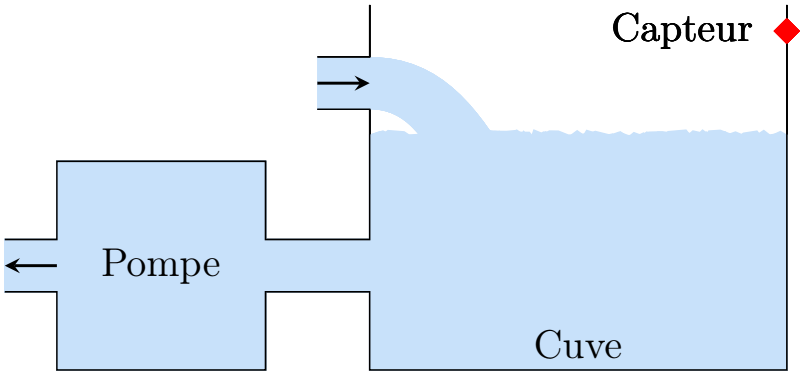


FIGURE 2.2. – Système victime de l'attaque Maroochy Shire

Du liquide remplit constamment la cuve que la pompe vide périodiquement. La pompe est contrôlée par une variable booléenne Pompe.Etat qui décrit si la pompe fonctionne ou non. Un capteur de niveau se trouve en haut de la cuve et est contrôlé par une variable booléenne Cuve.Niveau. On souhaite garantir que la pompe ne peut pas être arrêtée par le SCADA si le capteur de niveau en haut de la cuve détecte du liquide. Plus formellement on doit refuser une commande écrivant *false* sur Pompe.Etat si Cuve.Niveau vaut *true*.

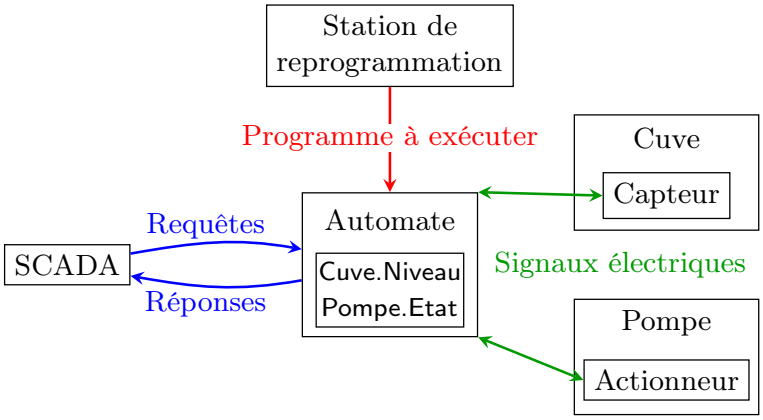


FIGURE 2.3. – Vue réseau du système

La figure 2.3 présente une vue du réseau contrôlant le système de façon similaire à la figure 2.1. Un SCADA communique avec un automate programmable qui est relié à l'actionneur de la pompe et au capteur de niveau. L'automate est donc un serveur où le SCADA viendra se connecter pour envoyer des commandes de lecture et écriture (représentées par une flèche bleue →). L'automate communique avec le capteur de la cuve et l'actionneur de la pompe via des signaux électriques ou un bus de terrain (représentés par une flèche verte →). Enfin, une station de programmation vient reprogrammer l'automate au besoin en lui envoyant le nouveau programme à exécuter (représenté par une flèche rouge →). Le SCADA peut donc par exemple faire une requête de lecture sur le capteur de niveau à laquelle l'automate répondra par la valeur la plus récente de la variable Cuve.Niveau qu'il a en mémoire. Dans notre exemple, si le capteur de niveau vaut *true*, l'opérateur voudra démarrer la pompe en écrivant *true* sur la variable Pompe.Etat. Enfin l'automate confirmera qu'il a bien appliqué l'écriture (par exemple en renvoyant la valeur écrite). Ces interactions sont décrites en figure 2.4.

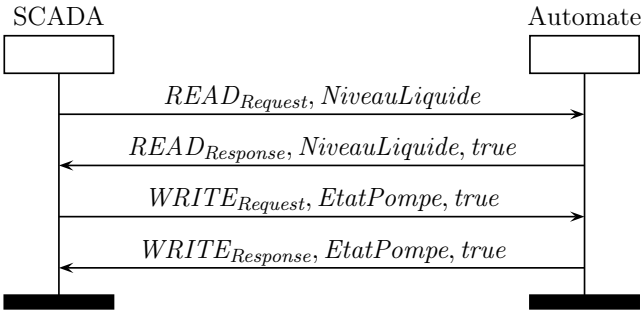


FIGURE 2.4. – Exemple de commandes

2.2. Spécificités des protocoles de communication industriels

Cette section décrit les particularités des protocoles de communication industriels par rapport à des protocoles de communication plus classiques comme SSH. La section 2.2.1 commence par expliquer la différence entre des protocoles d'établissement de session (*handshake*) et des protocoles de transport. Ensuite la section 2.2.2 décrit de façon générale les messages échangés via des protocoles de transport. Enfin la section 2.2.3 se focalise sur les protocoles qui seront étudiés dans cette thèse.

2.2.1. Protocoles de handshake et protocoles de transport

Les protocoles de communication se découpent généralement en plusieurs (séquences de) sous-protocoles. Un premier sous-protocole vise à établir une session entre le client et le serveur. Selon la sécurité proposée par le protocole, c'est aussi dans cette première étape que seront négociées les primitives cryptographiques utilisées et les clés générées. On désigne souvent ces sous-protocoles via le terme *handshake* (poignée de main). Une fois cette session ouverte, un sous-protocole dit de *transport* se charge d'envoyer les messages contenant le contenu applicatif (ou charge, ou encore *payload*). Cette distinction entre protocoles de handshake et de transport a des répercutions dans plusieurs domaines. La principale différence entre ces deux types de protocoles dans notre contexte est que les protocoles de handshake sont une séquence fixée d'actions visant à garantir des propriétés de sécurité. A l'opposé, les protocoles de transport ont pour but d'envoyer un message d'un correspondant vers l'autre (par exemple une commande en SSH). Cet envoi sera répété autant de fois que nécessaire et ne se terminera qu'à la demande du client ou du serveur (par exemple quand le client a envoyé toutes les commandes qu'il souhaitait). Ainsi, dans le protocole OPC-UA, le handshake est effectué une fois à l'ouverture de la session puis le protocole de transport est utilisé à chaque fois que le client envoie une commande au serveur. Une commande spéciale permet au client de se déconnecter et clôt le protocole. Par la suite, dans les chapitres 3 et 5, on considère que les sessions sont déjà établies et on se concentre sur les protocoles de transport. Dans le chapitre 4, on s'intéresse autant aux protocoles de handshake qu'aux protocoles de transport.

2.2.2. Messages envoyés par les protocoles de transport

Comme décrit en sections 2.1 et 2.2.1, les protocoles de transports sont utilisés par les clients pour transmettre leurs commandes aux serveurs. Les automates programmables (au niveau 1 de la classification de Purdue décrit en figure 1.1) ont des variables qui représentent l'état des capteurs et des actionneurs. Ces derniers interagissent avec le monde physique (au niveau 0). En particulier, les capteurs feront des mesures (par exemple de courant, de pression, etc.) et inscriront les valeurs mesurées dans les variables qui leur sont affectées. De façon analogue, les actionneurs agiront sur le monde physique en fonction des valeurs de variables dédiées. Les automates programmables agissent comme des serveurs avec les SCADA pour clients et leur permettent de lire et d'écrire sur leurs variables à l'aide de protocoles de transports. Les clients feront des

requêtes de lecture et d'écriture. Les réponses des serveurs seront constituées des valeurs des variables dont les clients ont demandé la lecture ou bien les confirmations de la prise en compte des requêtes d'écriture. Si ce modèle de communication s'applique particulièrement aux SCADA et aux automates, il est parfois transposables aux communications entre d'autres niveaux (les SCADA étant par exemple serveurs et les MES leurs clients).

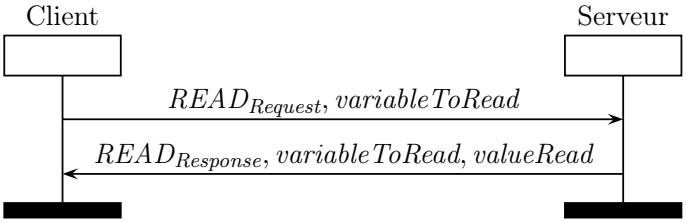


FIGURE 2.5. – Exemple de commande de lecture

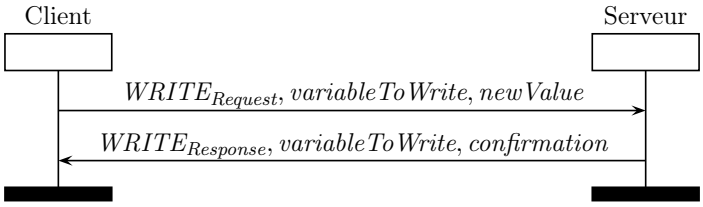


FIGURE 2.6. – Exemple de commande d'écriture

Les figures 2.5 et 2.6 montrent respectivement une commande de lecture et d'écriture de variable. Elles sont chacune accompagnées de la réponse du serveur. Dans le cas d'une lecture, le client précise la valeur qu'il souhaite lire et le serveur lui répond avec sa valeur courante. Lors d'une écriture, le client envoie la variable qu'il souhaite modifier et la nouvelle valeur qu'il veut lui donner et le serveur confirme le succès de l'écriture. Un numéro de séquence est généralement associé au message afin que le serveur et le client puissent s'accorder sur l'ordre et éviter des inversions. D'autres champs peuvent également apparaître en fonction du protocole utilisé.

Certains protocoles permettent d'envoyer plusieurs ordres de lectures et d'écritures en même temps. Par la suite, nous allons différencier les termes « message » et « commande » de la façon suivante :

Définition 1. *Une commande est un ordre atomique de lecture ou d'écriture par le client sur une variable d'un serveur (réciproquement pour la réponse du serveur). Un message représente une ou plusieurs commandes envoyées en même temps (du point de vue du protocole de communication).*

2.2.3. Détails des protocoles utilisés dans cette thèse

Il existe un multitude de protocoles de communication industriels avec à chaque fois un sous-protocole de transport. C'est le cas des protocoles cités en section 1.1.1 comme par exemple : MODBUS, CAN bus, Profibus, BACnet, OPC-UA, S7, DNP3. De même, de nombreux protocoles initialement non industriels sont utilisés dans l'industrie comme par exemple : FTP, FTPS, SSH ou encore HTTPS. Cette section vise à présenter les protocoles qui seront étudiés dans le reste de cette thèse : la section 2.2.3.1 présente le protocole MODBUS et la section 2.2.3.2 décrit le protocole OPC-UA.

2.2.3.1. MODBUS

MODBUS [MODBUS, 2004] est un protocole de communication industriel créé en 1979 par Modicon (maintenant Schneider Electric) in 1979. Devenu l'un des protocoles les plus populaires du domaine, il peut être utilisé soit sur un bus série soit via le protocole TCP. Dans le cadre de cette thèse, nous nous intéressons uniquement à la version TCP. En effet, cette dernière est plus récente que la version série qui tombe progressivement en désuétude, dû au fait que les systèmes industriels sont de plus en plus connectés via des réseaux Ethernet et IP. Quelque soit la version du protocole, les échanges sont toujours initiés par le client qui envoie des requêtes (de lecture ou d'écriture). Cela signifie que le serveur n'envoie jamais de messages de lui-même.

MODBUS ne manipule que deux types de données : les booléens et les entiers non signés sur 16 bits (Uint16). Ces deux types existent en lecture seule et en lecture/écriture. Le protocole possède ainsi quatre types, présentés en table 2.1. Le type de données Discrete Input (resp. Input Register) correspond donc à un booléen (resp. un entier) en lecture seule tandis que le type Coil (resp. Holding Register) est accessible en lecture et écriture. Toutes les variables sont adressées par un identifiant numérique (pouvant être vu comme un indice dans un tableau représentant l'ensemble des variables).

	Booléen	Uint16
Lecture seule	Discrete Input	Input Register
Lecture/écriture	Coil	Holding Register

TABLE 2.1. – Types de données du protocole MODBUS

En plus du numéro de séquence TCP, MODBUS introduit son propre compteur pour maintenir l'ordre des commandes. Ce dernier est appelé *transaction identifier* et est incrémenté de un à chaque requête du client (cela signifie que la réponse du serveur porte le même *transaction identifier*). D'autres paramètres sont présents dans l'entête MODBUS : (i) le *protocole identifier* est utilisé par compatibilité avec les versions non TCP, (ii) la taille du message (en octets), et (iii) un *unit identifier* qui sert à répartir les commandes sur les automates programmables reliés au serveur dans le cas où il y en a plusieurs. Le protocole rend possible la lecture et/ou l'écriture d'une plage de variables contiguës en une seule commande. Dans ce cas, le client spécifie la première variable à lire/écrire ainsi que leur nombre (par exemple ($READ_{Request}$, 5, 3) lira les variables 5, 6 et 7). Dans le cas de lectures, le serveur renvoie une liste de valeurs dans l'ordre de leur identifiants. Dans le cas des écritures, le serveur renvoie le nombre de registres écrits

avec succès. La figure 2.7 montre une lecture de variables suivie par une écriture de variables, toutes deux accompagnées par leurs réponses respectives avec n le *transaction identifier* et *ph* qui désigne les 3 champs *protocol identifier*, *taille du message* et *unit identifier* de l'entête.

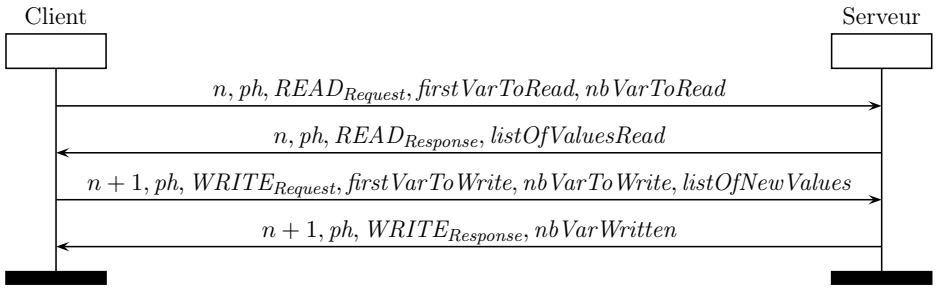


FIGURE 2.7. – Deux requêtes et réponses MODBUS

2.2.3.2. OPC-UA

OPC-UA, créé en 2006 puis standardisé en 2015 [IEC-62541, 2015], est l'un des protocoles de communication industriels les plus récents. Développé par la Fondation OPC [1], il est souvent considéré comme la prochaine référence des communications industrielles. C'est un protocole à plusieurs niveaux, allant de la couche transport au niveau applicatif. Une première nouveauté apportée par OPC-UA est d'inclure nativement une couche sécurité. Celle-ci implémente un protocole de handshake pour que les clients et les serveurs puissent générer des clés. Le client est alors invité à fournir un moyen d'authentification tel qu'un mot de passe ou un certificat en utilisant les clés fraîchement générées. L'utilisateur négocie par ailleurs un *mode de sécurité* parmi ceux proposés par le serveur. Ce mode de sécurité indiquera si les communications à venir seront signées, chiffrées ou en clair. Plus précisément les modes de sécurité d'OPC-UA sont :

— *None* qui n'apporte aucune sécurité (par compatibilité avec les matériels trop anciens ou ayant une trop faible puissance pour faire de la cryptographie) ;

— *Sign* où les messages sont signés mais envoyés en clair ;

— *SignEncrypt* où les messages sont signés et chiffrés.

Ensuite, le sous-protocole de transport est utilisé par les clients pour envoyer des commandes aux serveurs en accord avec le mode de sécurité convenu. D'après l'équipe de support de la Fondation OPC [2], le sous-protocole de transport d'OPC-UA vient se mapper sur les couches 5 et 6 du modèle OSI (Session et Présentation). Les commandes envoyées ensuite via le protocole de transport se placent dans la couche 7 (Application). Certains travaux non liés à la Fondation OPC affirment quand à eux que la pile tout entière vient se mapper sur la couche applicative du modèle OSI [Post et al., 2009].

1. Un consortium des principaux acteurs du domaines.
2. http://forum.unified-automation.com/topic1488.html

L'autre nouveauté majeure apportée par OPC-UA porte sur les types de variables pouvant être utilisés. Ces types incluent les booléens, les entiers (signés ou non) de différentes tailles, les flottants, ainsi que des types plus complexes comme les chaînes de caractères, les dates et les heures ou encore la prise en compte des langues. Ces types peuvent par ailleurs être agrégés en types plus complexes comme des tableaux (multidimensionnels) ou encore des structures arbitraires. Les variables de ces différents types sont agencées en un *espace d'adressage*. Contrairement au protocole MODBUS où les variables sont essentiellement indépendantes les unes des autres, en OPC-UA, cet espace d'adressage est organisé sous forme d'un graphe potentiellement cyclique. Les variables y sont appelées des nœuds et un nœud peut contenir d'autres nœuds. Les espaces d'adressage peuvent être spécifiés dans des fichiers au format XML, comme spécifié dans la documentation du protocole [IEC-62541, 2015]. Cette fonctionnalité s'avère extrêmement pratique afin de modifier aisément l'espace d'adressage d'un serveur et de les partager d'un serveur à l'autre. Il peut par exemple être vendu avec un composant pour spécifier ses variables.

Dans le cas de la couche transport d'OPC-UA, les commandes comme Read et Write sont appelés des services. En particulier, Read et Write sont très similaires à MODBUS. OPC-UA introduit le service Browse qui permet de lister les fils d'un nœud de l'espace d'adressage. Ce service est utile dans le cas de clients « interactifs » où l'utilisateur se déplace dans l'arborescence des nœuds comme il se déplacerait dans les dossiers de son disque dur. D'autres services propres à OPC-UA permettent par exemple au client de « s'abonner » à des variables et de recevoir leur valeur courante à intervalles réguliers. Plusieurs champs sont ajoutés par OPC-UA en plus des commandes :

— *mh* : le *message header* contenant des informations comme la taille du message.

— *sh* : le *security header* contenant un identifiant aléatoire appelé *security token*.

— n : le *sequence number* qui est incrémenté afin de maintenir l'ordre des commandes. Similaire au *transaction identifier* en MODBUS, à la différence qu'il est incrémenté à chaque requête **et** à chaque réponse.

— rID_i : L'identifiant de la requête afin d'y associer correctement les réponses.

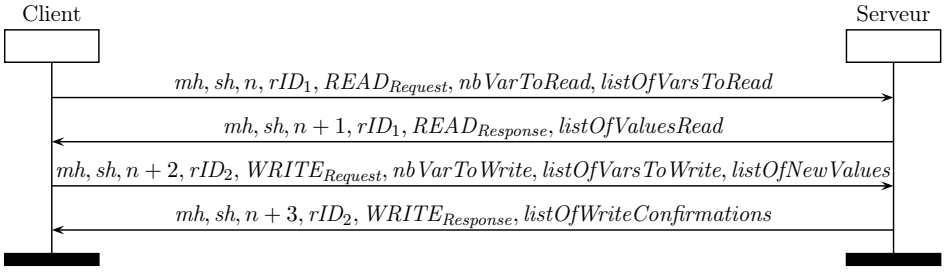


FIGURE 2.8. – Deux requêtes et réponses OPC-UA

En fonction du mode de sécurité utilisé, une signature, et si besoin un padding, seront ajoutés. Similairement à la figure 2.7 pour MODBUS, la figure 2.8 montre une lecture

de variables suivie par une écriture de variables, toutes deux accompagnées par leurs réponses respectives. Ici, comme l'espace d'adressage n'est pas contiguë, le client envoie explicitement la liste des variables à lire ou écrire.

2.3. Objectifs de la thèse

Comme présenté en section 1.4, plusieurs attaques sur des systèmes industriels ont déjà eu lieu par le passé. Il serait bien hasardeux de prétendre que les contributions de cette thèse auraient pu empêcher ces attaques. En effet, ajouter des protections ne sert en général qu'à rendre la tâche de l'attaquant plus ardue et le forçant à prendre un autre chemin et si possible le décourager de passer à l'acte. Cependant, un attaquant motivé et ayant du temps, des moyens et de la compétence (en particulier dans le cas de Stuxnet) arrivera à ses fins quels que soient les protections mises en place. Ainsi, on se propose ici de comprendre quelles étaient les vulnérabilités mises en jeux dans les attaques montrées en section 1.4 et de décrire comment celles-ci pourraient être comblées par les contributions de cette thèse. Pour cela nous allons prendre l'exemple de l'attaque Maroochy Shire comme fil rouge. Pour rappel, les vulnérabilités pouvant être imputées à cette attaque sont :

1. l'absence de mécanisme de sûreté empêchant de répandre le contenu de la cuve,
2. l'absence d'authentification dans les protocoles de communication,
3. l'absence de préparation des employés à une attaque.

Dans la suite de cette section, nous allons présenter les trois chapitres de contribution de cette thèse, chacun relatif à l'une des vulnérabilités énumérées ci-dessus. Le chapitre 3 présente une dispositif de filtrage pour les systèmes industriels. Ensuite, le chapitre 4 s'intéresse à la vérification de protocoles cryptographiques dans le contexte des systèmes industriels. Enfin, le chapitre 5 propose une méthode de production de scénarios d'attaques contre des propriétés de sûreté.

2.3.1. Le projet ARAMIS : un dispositif de filtrage

Dans le cas de la vulnérabilité 1, il parait évident que rien n'a empêché les pompes de s'arrêter alors qu'elles auraient dû fonctionner. Comme décrit en annexe A.3, plusieurs possibilités s'offrent à l'attaquant pour arriver à cette éventualité. La pompe aurait par exemple pu démarrer automatiquement puis être stoppée par l'attaquant via une commande usurpant le SCADA. De façon similaire, un capteur contrôlant le niveau de liquide aurait pu être forcé par l'attaquant pour faire croire aux pompes que la cuve est vide. L'attaquant aurait aussi pu reprogrammer l'automate contrôlant la cuve afin que la pompe ne démarre pas du tout. Pour répondre à ces attaques, le projet ARAMIS [ARAMIS, 2014] vise à proposer un dispositif permettant de cloisonner physiquement les réseaux des systèmes industriels et de filtrer les échanges en tenant compte des contraintes métier. Ce dispositif devra ainsi rejeter tout flux identifié comme interdit (liste noire) ou non identifié comme autorisé (liste blanche), donc potentiellement malveillant. Étant un dispositif embarqué, il doit respecter des contraintes de mémoire en plus des contraintes de temps des communications industrielles décrites en section 1.1.1.

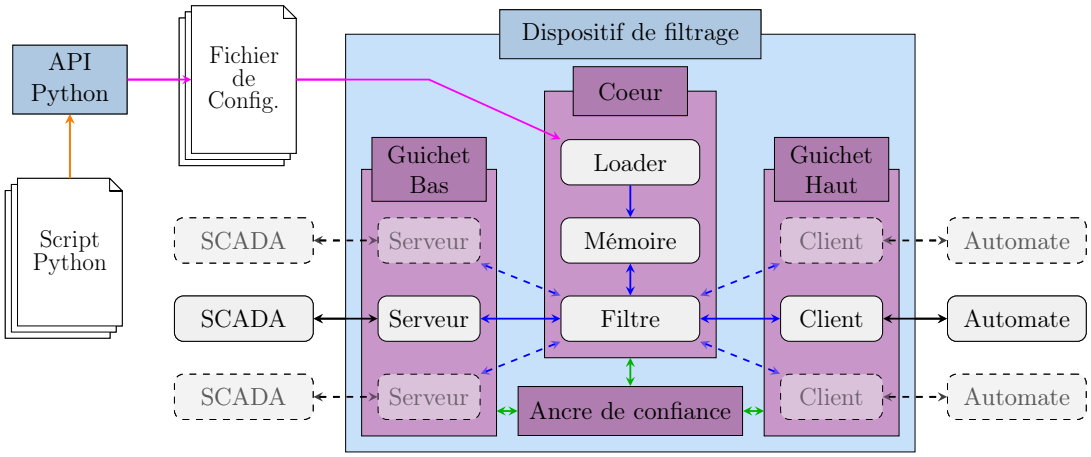


FIGURE 2.9. – Composants du dispositif de filtrage

La figure 2.9 présente une vue simplifiée d'un prototype du dispositif ARAMIS. Ce prototype résulte de réflexions entre les membres du projet ainsi que d'idées internes à cette thèse et ne décrit pas le produit issu du projet. Ainsi par la suite, nous désignerons ce dispositif en tant que « dispositif de filtrage » afin d'éviter toute confusion. Le dispositif est divisé en trois systèmes sur puce – *System On Chip* (SOC) – distincts, chacun avec ses propres processeurs et mémoires, ceci afin de garantir une isolation matérielle. Deux *guichets*, haut et bas (respectivement pour haute sécurité et basse sécurité) entourent le *coeur* du dispositif sur lequel est effectué le filtrage. Vient s'ajouter une *ancre de confiance* qui sert de *Hardware Security Module* (HSM) dans le dispositif pour les opérations cryptographiques (représentées par des flèches vertes ↔). Ainsi, en plus d'assurer les communications sécurisées pour les protocoles de communication qui le permettent (par exemple OPC-UA), l'ancre permet de chiffrer les journaux émis par le dispositif. Elle pourrait également assurer l'authenticité et l'intégrité des fichiers de configuration du filtre en vérifiant une signature.

Les clients SCADA voulant interagir avec les automates programmables envoient des requêtes qui sont interceptées par le dispositif de filtrage. Elles sont alors désencapsulées, c'est à dire converties dans un format intermédiaire commun à tous les protocoles supportés par le dispositif. Il sera ainsi possible d'appliquer du filtrage sur tous les messages quelque soit leur protocole de communication. Si le filtre accepte le message, il sera transmis de l'autre côté du dispositif. Hors ligne, la configuration du dispositif et le chargement des règles de filtrage sont effectués au moyen d'un langage textuel dit « bas niveau » (représenté par une flèche rose →). Ce fichier est lu par un module de chargement, le *loader* qui stocke ces informations en mémoire afin qu'elles puissent être accédées par le filtre. Pour faciliter et automatiser partiellement la configuration du dispositif, une API permet de générer les fichiers en langage bas niveau à partir de scripts Python (représenté par une flèche orange →).

Ainsi, via le dispositif de filtrage, une commande d'arrêt de la pompe pourrait être bloquée si le capteur de niveau indique que la cuve est pleine. Les écritures sur ce capteur pourraient par ailleurs être bloquées afin qu'il ne soit pas forcé pour perturber la pompe. En chapitre 3, nous nous focaliserons sur le fonctionnement du filtre, les propriétés qu'il

peut vérifier et les langages utilisés pour sa configuration. En particulier, nos travaux sur les langages de configuration ont donné lieu à une publication à la conférence CRITIS 2016 [Puys et al., 2016c] et ceux sur l'algorithme de filtrage à la conférence WCICSS 2017 [Badrignans et al., 2017].

2.3.2. Vérification formelle de protocoles de communication industriels

Dans le cas de la vulnérabilité 2, les automates programmables utilisés durant l'attaque Maroochy Shire communiquaient via les protocoles de communication industriels MODBUS et DNP3. Ces deux protocoles, comme beaucoup d'autres du domaine, ne fournissent aucune protection contre des attaquants. Un attaquant capable de communiquer avec les automates peut donc lancer arbitrairement des commandes affectant le procédé. En particulier, il pourrait se faire passer pour le client et lancer une commande afin d'arrêter la pompe lorsque la cuve est pleine. Pour pallier ce risque, plusieurs travaux académiques ont proposé d'ajouter de la sécurité à MODBUS. Le protocole OPC-UA a notamment été développé à partir de son prédécesseur OPC et implémente des mesures de sécurité prétendues à l'état de l'art des attaques. Cependant, comment s'assurer que les solutions proposées peuvent réellement contribuer à empêcher les attaques ? La vérification de protocoles cryptographiques consiste à appliquer des méthodes formelles sur la spécification de tels protocoles afin de tester des propriétés.

Depuis 20 ans, de nombreux travaux ont contribué à la vérification de protocoles cryptographiques comme SSH et TLS. Cependant très peu de vérifications ont été menées sur les protocoles industriels. Il parait difficile à accepter que des protocoles contrôlant des systèmes aussi critiques que des centrales nucléaires ou des barrages puissent n'avoir jamais été vérifiés formellement. Le chapitre 4 s'intéresse ainsi à porter et étendre les avancées de ce domaine aux protocoles de communication industriels. Pour cela, nous proposons une vérification formelle du handshake d'OPC-UA à l'aide de l'outil ProVerif en modélisant le protocole à partir du standard. Ceci a donné lieu à une publication à la conférence SAFECOMP 2016 [Puys et al., 2016a]. Par ailleurs, nous proposons une formalisation de propriétés basées sur l'ordre des messages, actuellement non supportées par les outils de vérification. Cette propriété est cruciale pour les systèmes industriels car réordonner les messages d'une trace d'exécution respectant des propriétés peut mener à violer ces dernières. Ce travail a donné lieu à une publication à la conférence SECRYPT 2017 [Dreier et al., 2017b]. Nous appelons cette classe de propriété *intégrité du flux* et l'implémentons dans l'outil Tamarin.

2.3.3. Recherche automatique de scénarios d'attaques applicatives

Dans le cas de la vulnérabilité 3, les employés n'étaient pas préparés à l'attaque qui est survenue. On peut aisément s'en rendre compte du fait que l'attaque s'est déroulée sur plusieurs mois et que les conséquences étaient visibles dès les premières actions mais non détectées. Afin de préparer au mieux les opérateurs, il convient au préalable d'identifier les faiblesses du système. Pour cela, plusieurs méthodes d'analyse de risques ont été proposées au fil des années. Certaines, comme EBIOS [ANSSI, 2010] et MEHARI [CLUSIF, 2010] en France, visent à identifier les risques en matière de sécurité pesant sur les systèmes d'information. A l'opposé, d'autres méthodes comme HA-

ZOP [Kletz, 1999, IEC-61882, 2001] et AMDEC [IEC-60812, 1985] évaluent les risques en matière de sûreté. A notre connaissance, ces approches sont souvent très spécifiques à la sécurité ou bien à la sûreté. Elles sont aussi généralement très structurées mais souvent manuelles. Aussi, plusieurs outils ont été développés pour aider à les appliquer. Une autre approche consiste à utiliser des techniques de vérification afin de s'assurer qu'un système garantit certaines propriétés. Une multitude de travaux ont été proposés pour effectuer de telles analyses, tels que la vérification de modèles [Emerson and Clarke, 1980, Queille and Sifakis, 1982] – *model-checking* –, les réseaux de Petri [Murata, 1989], la satisfiabilité modulo des théories [Barrett et al., 2009] – *SMT solving* – ou encore la preuve de théorèmes [Paulson, 1994, Bertot and Castéran, 2013]. Ces approches sont par contre automatiques ou à défaut semi-automatiques. Une fois que l'utilisateur a décrit le système et les propriétés, l'outil est capable de conclure automatiquement.

Pour répondre à la problématique de la préparation aux attaques, nous proposons l'approche A^{2}SPICS, pour *Applicative Attack Scenarios Production for Industrial Control Systems*. Cette méthode d'analyse consiste à utiliser des techniques de vérification par simulation telles que le model-checking afin de vérifier si un système assure des propriétés de sûreté en présence d'attaquants. La figure 2.10 illustre l'idée globale de l'approche. Afin de prendre en compte à la fois sécurité et sûreté, A^{2}SPICS fonctionne en deux phases, chacune reposant sur une analyse de risques. Dans une première phase, on s'intéresse à une analyse de risques en terme de sécurité basée sur la topologie du réseau, sur les objectifs des attaquants potentiels, ainsi que sur les fonctionnalités de sécurité supportées par les protocoles de communication. A l'issue de cette analyse, des *modèles d'attaquants* sont produits. Ces modèles détaillent notamment les positions, capacités et objectifs des attaquants. Cette première partie de l'approche a fait l'objet d'un article à la conférence française AFADL 2016 [Puys et al., 2016b].

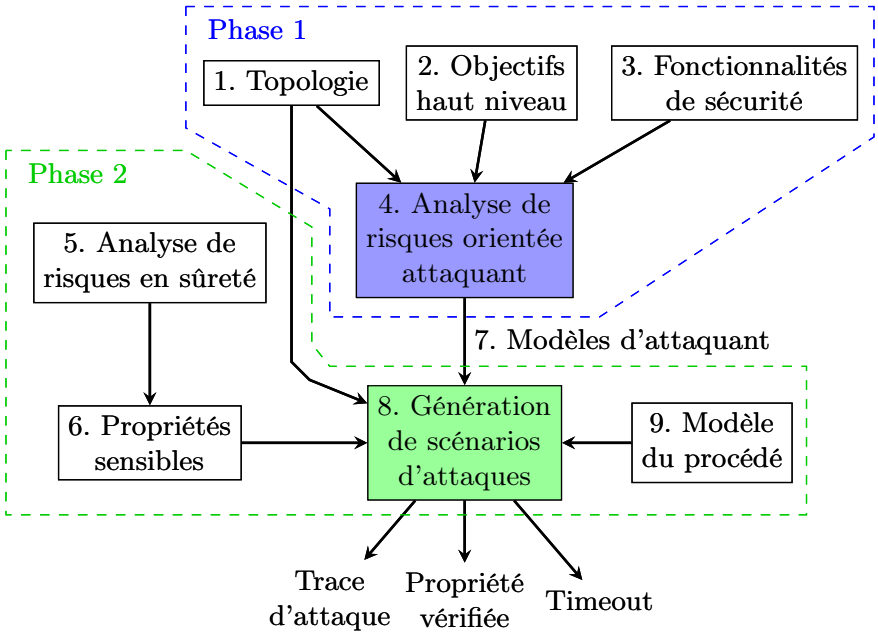


FIGURE 2.10. – L'approche A^{2}SPICS

Dans une seconde phase, on tire avantage du fait que les systèmes industriels sont très bien analysés en terme de sûreté de fonctionnement. Aidés par les propriétés à garantir, résultant de ces analyses de sûreté, on s'intéresse à voir si les modèles d'attaquants produits par la phase 1 sont en mesure de violer les propriétés de sûreté de fonctionnement. Pour cela, on prend en compte le comportement du procédé ainsi que les variables qu'il manipule dans la modélisation. Par exemple dans le cas d'un haut fourneau, une propriété à vérifier serait que le four ne démarre pas si la porte est ouverte. Via une méthode de vérification, on peut alors savoir, pour chaque propriété, si un attaquant peut la violer et, le cas échéant, obtenir le chemin menant à cette attaque.

En principe, l'approche n'est pas dépendante de la méthode de vérification choisie. Ces méthodes faisant l'objet de recherches depuis des décennies, divers outils existent avec chacun leur paradigme de modélisation. On peut alors se poser la question de la « classe d'outils » à utiliser. Historiquement, les outils de vérification ont été utilisés pour vérifier des propriétés de sûreté de fonctionnement. A notre connaissance, ce n'est qu'en 1998 que Lowe présente Casper/FDR [Lowe, 1998], un outil permettant de vérifier des protocoles cryptographiques en présence de l'intrus Dolev-Yao [Dolev and Yao, 1981] à l'aide du model-checker FDR [Roscoe, 1994, Gibson-Robinson et al., 2014]. Casper/FDR génère un modèle vérifiable par FDR incluant le protocole à vérifier et les règles de déduction de l'intrus. Ces travaux ont mené à la définition d'une nouvelle classe d'outils, les outils de vérification de protocoles cryptographiques, optimisés pour des propriétés de sécurité.

Afin d'implémenter l'approche A^{2}SPICS, on peut alors se demander quelle classe d'outils utiliser ? Il parait en effet tentant de profiter du fait que les outils de vérification de protocoles simulent d'eux-mêmes le rôle de l'attaquant. D'un autre côté, ils sont optimisés pour des propriétés de sécurité alors que nous souhaitons tester des propriétés de sûreté. Les outils « classiques » seraient alors peut être plus adaptés. En chapitre 5, nous détaillons le fonctionnement de l'approche A^{2}SPICS et nous l'appliquons sur une étude de cas. Nous discutons ensuite des choix d'implémentations sur les classes d'outils nous semblant les mieux taillés pour des vérifications de propriétés de sûreté en présence d'attaquants.

Deuxième partie

Contributions

Chapitre 3

Filtrage dans les systèmes industriels

> Computers are good at following instructions, but not at reading your mind.
>
> *(Donald Knuth, 1984)*

Résumé du chapitre

La sécurité des systèmes industriels peut être en partie assurée par des dispositifs de filtrage. Permettant une compartimentation du système, ils sont aussi en mesure de bloquer les flux ne respectant pas une politique de sécurité. Nous débutons ce chapitre par quelques références sur le filtrage applicatif, ses applications aux systèmes industriels et nous discutons des propriétés pouvant être attendues par un filtre sur ces systèmes. Par la suite, nous proposons un filtre applicatif dédié aux systèmes industriels. Implémenté au sein d'un dispositif de rupture, il est en mesure de vérifier des règles classiques sur les droits d'accès des clients aux variables des serveurs. Il est également capable de garantir des propriétés orientées métier contrôlant notamment les valeurs possibles des variables, la temporisation des commandes et l'état global du système. Nous proposons une API Python permettant de générer automatiquement des fichiers de configuration lisibles par le filtre. Nous montrons comment configurer le filtre dans le cas l'attaque Maroochy Shire. Enfin nous discutons du positionnement de ce filtre par rapport à l'état de l'art et des ses généralisations possibles.

Sommaire

3.1. Contexte

Nous avons décrit en sections 1.4 et 2.3.1 l'attaque Maroochy Shire où l'attaquant est parvenu à répandre le contenu d'une cuve d'eaux usagées. Il a pour cela empêché des pompes de fonctionner lorsqu'elles auraient dû. Plusieurs vulnérabilités pouvaient être imputées à cette attaque et notamment l'absence de mécanisme de sûreté empêchant de répandre le contenu de la cuve. Comme décrit en annexe A.3, plusieurs possibilités s'offrent à l'attaquant pour déclencher cette situation. La pompe aurait par exemple pu démarrer automatiquement puis être stoppée par l'attaquant via une commande usurpant le SCADA. De façon similaire, si une variable est associée à un capteur contrôlant le niveau de liquide, cette variable devrait être en lecture seule pour le client. Cependant dans la pratique il est fréquent de voir toutes les variables d'un automate programmable en mode lecture et écriture (soit parce que le mode lecture seul n'est pas possible, soit par manque de sensibilisation de celui qui le programme). Dans cette situation on pourrait imaginer l'attaquant réécrire cette variable de façon à faire croire que la cuve n'est pas pleine. Il aurait aussi pu reprogrammer l'automate contrôlant la cuve afin que la pompe ne démarre pas du tout.

La protection des systèmes industriels contre ce type d'attaques peut être en partie assurée par l'ajout de mécanismes de filtrage. Ces filtres, matériels ou logiciels, reçoivent en entrée un flux de messages et vérifient s'ils sont conformes à une politique de sécurité établissant la configuration du filtre. Les filtres se distinguent des IDS – *Intrusion Detection Systems* – par le fait qu'ils obligent le respect de la politique de sécurité. Les filtres peuvent ainsi bloquer les messages violant cette propriété, là où les IDS feront remonter des alertes qui pourront ensuite être corrélées avec d'autres informations pour une prise de décision ultérieure. Ils se distinguent aussi des diodes qui ont pour utilité de ne laisser passer les messages que dans un sens (autorisant éventuellement les acquittements de messages) mais qui ne prévoient aucun filtrage. On s'intéresse ici à un filtre qui se trouverait entre deux niveaux consécutifs de la classification de Purdue (présentée

en figure 1.1 du chapitre 1) afin de les cloisonner. Ainsi les organes du niveau N seraient des clients pour des serveurs du niveau $N-1$, comme présenté en figure 3.1. Par exemple dans les communications entre le niveau 3 (management de la production) et le niveau 2 (supervision et contrôle), les clients seront les MES et les serveurs seront les SCADA. De la même façon, entre le niveau 2 et le niveau 1 (traitement de l'automatisme), les clients seront les SCADA et les serveurs seront des automates programmables.

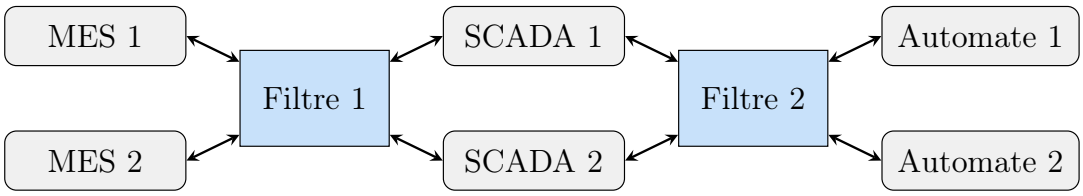


FIGURE 3.1. – Positions possibles d'un filtre

On notera également que le dispositif de filtrage présenté ici est un filtre applicatif dans le sens où il vérifiera les messages en tenant compte de leurs *payloads*, c'est à dire dans le contexte des systèmes industriels des commandes envoyées. A l'opposé, un filtre réseau ne regardera que les entêtes des messages afin de s'assurer par exemple que les correspondants sont autorisés à parler entre-eux. Autant les filtres applicatifs que réseaux peuvent être à états. Dans ce cas, les premiers garderont une copie locale de l'état du système industriel (par exemple des valeurs des variables) tandis que les seconds mémoriseront par exemple les connexions ouvertes sans se focaliser sur leur contenu.

3.1.1. Quelques références sur le filtrage applicatif

Les travaux sur la formalisation du filtrage applicatif de messages sont intrinsèquement liés à la définition des propriétés de sûreté et de vivacité. Cette différentiation a été initiée par Lamport en 1977 [Lamport, 1977] où il définit que la sûreté garantit que quelque chose – *de mauvais* – ne doit pas arriver (par exemple un attaquant ne doit pas apprendre la clé secrète). Plus formellement, lorsqu'un préfixe d'une trace d'exécution viole la propriété alors toutes les traces construites à partir de ce préfixe la violent également. Autrement dit, une fois le système entré dans un mauvais comportement, il n'y a plus de retour en arrière. A l'opposé, la vivacité garantit que quelque chose – *de bien* – doit arriver un jour (par exemple si un programme ouvre un fichier alors il doit finir par le refermer). En 1985, Alford, Lamport et Mullery formalisent la notion de sûreté dans [Mack W. Alford, 1985], tandis que Alpern et Schneider formalisent celle de vivacité dans [Alpern and Schneider, 1985]. En 1993, Chang *et al.* [Chang et al., 1993] proposent une classification alternative à la classique sûreté/vivacité. Cette classification propose une granularité plus fine des propriétés de vivacité et ayant la particularité d'être une hiérarchie et non une partition. Le concept de *Security Automata* est introduit sans le nommer par Alpern et Schneider en 1987 [Alpern and Schneider, 1987] puis formalisé en 2000 par Schneider [Schneider, 2000]. Il définit alors ces automates comme une classe d'automates de Büchi [Eilenberg and Tilson, 1974] garantissant des propriétés de sûreté et permettant de vérifier une politique de sécurité dans un système, propriété qu'il nomme *Execution Monitoring*.

Plusieurs travaux ont ensuite raffiné les propriétés pouvant être filtrées et ont proposé de nouveaux mécanismes de vérification [Viswanathan and Kim, 2004, Pnueli and Zaks, 2006, Bauer et al., 2011]. Ces travaux sont des mécanismes dits de « *runtime monitoring* ». Pour une séquence d'entrées ils fournissent une séquence de verdicts pour des propriétés données. C'est donc par ces formalismes que peuvent être définis les systèmes de détection d'intrusion tels que Suricata [Suricata, 2016], Snort [Snort, 2016] ou Bro [Paxson, 1999]. Une liste plus complète de ces outils et leurs fonctionnalités a été établie par Debar *et al.* [Debar et al., 2000]. Par ailleurs ces systèmes sont de plus en plus utilisés dans le contexte des systèmes industriels. On peut notamment citer les travaux de Cheung *et al.* [Cheung et al., 2007], de Morris *et al.* [Morris et al., 2013] ou encore de Diallo et Feuillet [Diallo and Feuillet, 2014].

De façon analogue aux travaux de « *runtime monitoring* », le « *runtime enforcement* » vise à produire des sorties respectant des propriétés à garantir. Ainsi la sortie est le plus long préfixe correct de l'entrée respectant la propriété. La forme probablement la plus simple de *runtime enforcement* est un filtre classique qui laisse passer un message ou le bloque. L'idée derrière les *security automata* de Schneider est que le filtre est implémenté par un automate représentant les comportements autorisés. Ainsi, lorsque le filtre rencontre un message qu'il doit bloquer, l'automate ne trouve pas de transition liée au message. Il part donc dans un état absorbant et refuse tous les messages suivants (puisqu'une propriété de sûreté violée ne peut pas être « rattrapée »). Plusieurs travaux ont ensuite été introduits, proposant plusieurs algorithmes de filtrage [Fong, 2004, Ligatti et al., 2005, Guiochet and Powell, 2005, Falcone et al., 2008, Koucham et al., 2016], certains rendant par exemple possible de modifier les messages filtrés afin de toujours produire une sortie correcte (par exemple les tronquer). Cependant, ces algorithmes ne visent pas de catégories particulières de systèmes et semblent difficilement applicables au contexte des systèmes industriels. Ils permettent notamment d'exprimer des propriétés de vivacité (bornée ou non) en retardant des messages ce qui n'est pas envisageable pour des systèmes temps réel. Par ailleurs, aucun des mécanismes de filtrage présentés ci-dessus n'a été appliqué aux protocoles de communication industriels et ils ne sont pas optimisés pour le format de ces communications.

D'autres s'appuient sur les spécificités des systèmes industriels pour offrir un filtrage adapté à leurs besoins. En 2011, Cox [Cox, 2011] utilise des méthodes d'*automatic computing* pour filtrer des communications MODBUS. En modélisant le système industriel comme un système linéaire représenté par une fonction de transfert, il repère lorsque le procédé dévie de son état normal. Une fois ce comportement détecté, les accès au procédé sont bloqués pour l'isoler du reste du système. Toujours en 2011, Cárdenas *et al.* [Cárdenas et al., 2011] proposent une méthodologie similaire qu'ils mettent en application sur la modélisation d'un réacteur. Dans ces deux travaux, le filtrage est donc effectué sur la base de l'observation du procédé et non sur les commandes échangées. En 2014, Chen et Abdelwahed [Chen and Abdelwahed, 2014] présentent un mécanisme de filtrage applicatif de communications MODBUS reposant également sur des systèmes linéaires. En conservant un historique des valeurs de variables ils estiment les valeurs futures par des classificateurs bayésiens et repèrent ainsi si l'état du système industriel va varier de façon anormale. Une réponse est alors prise de façon automatique ou proposée à l'opérateur basée sur différents critères (coût, efficacité, impact sur les performances).

3.1.2. Propriétés attendues pour un filtre sur des systèmes industriels

Nous avons listé en section 1.1 des propriétés de sécurité souhaitées pour les systèmes industriels. Dans le domaine du filtrage applicatif pour les systèmes industriels, on considère un attaquant du côté client qui a accès aux règles configurées sur le filtre, ainsi qu'à leurs implémentations, mais qui ne peut pas les modifier. Cet attaquant peut par exemple être une machine venant s'ajouter au système industriel (quelqu'un physiquement présent sur le site qui brancherait sa machine à un câble). Il peut également s'agir d'une machine connue du système et qui serait compromise (soit infectée par un virus, soit pilotée par un opérateur malveillant). Ce type d'attaquant est en mesure d'intercepter, modifier, rejouer des messages qu'il voit passer ou de forger ses propres messages, similaire à l'intrus Dolev-Yao qui sera présenté en chapitre 4. Du point de vue du filtre, il est considéré comme n'importe quel client, envoyant des commandes potentiellement malveillantes vers un serveur situé de l'autre côté. Ainsi tout client, y compris l'attaquant, verra ses commandes filtrées et refusées si elles sont contraires à la configuration du filtre. Un filtre ne peut pas seul garantir les propriétés énoncées en section 1.1.2 pour le système tout entier, cependant il peut y contribuer de la façon suivante.

Disponibilité : Le filtre peut empêcher des dénis de services « applicatifs » au sens où il cherchera à éviter qu'une suite d'actions soit exécutée trop rapidement ou trop souvent pour le procédé. Par exemple, dans le cadre du projet Aurora présenté en section 1.4, des chercheurs du laboratoire national de l'Idaho ont envoyé des commandes d'ouverture et de fermeture aux disjoncteurs d'un générateur diesel. Ces ordres envoyés trop rapidement et de façon désynchronisée ont eu des conséquences physiques sur les mécanismes de rotation du générateur, provoquant sa destruction. Ce type d'attaque peut par exemple être évité en refusant les flux qui ne garantissent pas un certain délai entre les commandes d'ouverture et de fermeture.

Contrôle d'accès : Le filtre peut *a minima* faire le travail d'un pare-feu classique en n'autorisant les flux que d'émetteurs connus vers des destinataires connus. Cependant, un contrôle plus fin sur le contenu des commandes serait préférable en décrivant notamment à quelles variables un client peut accéder (par exemple définir sur chaque automate programmable des variables peu critiques accessibles par tous les opérateurs et des variables critiques accessibles uniquement par les administrateurs).

Sûreté de fonctionnement : Comme décrit en chapitre 1, la sûreté de fonctionnement est un enjeu bien connu des systèmes industriels et de nombreux travaux de recherche existent sur le sujet. Dans un contexte de sûreté, le système contrôlant le procédé est sûr en l'absence d'attaquant. Par exemple, un SCADA interdirait aux opérateurs de démarrer un four si sa porte est ouverte, en faisant périodiquement des lectures sur le capteur de la porte. Un attaquant n'étant pas soumis à ces restrictions, il peut envoyer une commande de démarrage au four indépendamment de l'état du capteur. Ainsi le filtre a pour tâche d'empêcher un attaquant d'outrepasser les mécanismes de protection existants destinés aux opérateurs. Cela nécessite que le filtre puisse vérifier des règles de sûreté de fonctionnement.

Traçabilité : Le filtre peut garantir la traçabilité en faisant remonter des informations sur les messages qu'il bloque et/ou laisse passer à un SIEM (*Security Information and Event Management*). Dans le cas de messages bloqués par des règles de sûreté de fonctionnement, le filtre devrait en plus garder une trace de la raison applicative pour laquelle le message a été bloqué (par exemple un délai trop court entre deux commandes).

Intégrité et non-répudiation : L'intégrité et la non-répudiation reposent essentiellement sur des fonctions de sécurité devant être implémentées au niveau des protocoles de communication, comme des signatures cryptographiques. Le filtre pourra par exemple vérifier ces signatures avant qu'elles arrivent à leurs destinataires. Le filtre pourrait aussi servir de passerelle entre des équipements. Un message non sécurisé serait signé voir chiffré par un filtre puis reçu et déchiffré par un autre filtre qui vérifierait la signature avant de faire suivre le message aux équipements. Les filtres agiraient donc comme des proxies ou des VPN. Ils pourraient même convertir les messages échangés en un protocole garantissant l'intégrité des communications, comme OPC-UA. Enfin, une autre notion d'intégrité que le filtre devrait assurer est l'intégrité de sa configuration. Cela pourrait par exemple passer par la signature du fichier de configuration, assurant que la configuration du filtre provient bien d'un opérateur connu.

3.1.3. Propriétés vérifiées par notre filtre

Pour répondre aux problématiques de filtrage énoncées en section 3.1.2, le consortium ARAMIS a retenu certaines propriétés comme objectifs pour le filtre d'un dispositif de rupture. Certaines sont issues de normes et référentiels tels que le guide de bonnes pratiques de l'ANSSI ou de l'IEC/CEI 62443 présentés en section 1.2 ; d'autres ont été sélectionnées par des experts de différents domaines pour répondre à des cas d'usage. Nous classons les propriétés vérifiées par le filtre en deux catégories. La première regroupe des fonctionnalités de base d'un pare-feu applicatif sur la topologie du réseau (déclaration des hôtes, permissions, etc.). Ces fonctionnalités peuvent être assimilées à du filtrage réseau. La seconde catégorie propose des fonctionnalités avancées liées au procédé et pouvant contribuer à assurer sa sûreté de fonctionnement. Ces dernières étant liées au procédé, elles sont dites « métier ». Ces propriétés relèvent pour la plupart de l'inspection des paquets en profondeur – *Deep Packet Inspection* (DPI) – afin d'analyser le contenu des paquets et la logique des commandes échangées.

Propriétés sur la topologie réseau :

— Assurer que seuls des protocoles autorisés et des services (lecture, écriture, etc.) autorisés pourront traverser le dispositif. Par exemple, si un protocole de communication n'a pas été explicitement autorisé, les messages envoyés par ce protocole seront systématiquement rejetés. De la même façon, si un service n'a pas été explicitement autorisé, les messages invoquant ce service seront rejetés. Cela permet par exemple d'avoir un dispositif en lecture seule (en n'autorisant pas le service d'écriture de variables).

— Assurer que seuls les correspondants autorisés communiquent entre eux. Cette propriété impose à un client de ne pouvoir faire des requêtes qu'à un certain

ensemble de serveurs, tout autre requête étant rejetée. Leur identification peut se faire grâce aux adresses IP ou par tout autre moyen mis à disposition par le protocole de communication utilisé (par exemple pour distinguer les humains contrôlant les clients).

— Assurer le respect de droits d'accès des clients aux variables. Cette propriété de contrôle d'accès impose à un client de ne pouvoir faire des requêtes qu'à un certain ensemble de variables, tout autre requête étant rejetée. Par exemple, un client avec peu de droits (un MES) ne pourrait accéder qu'à certaines variables non critiques tandis qu'un client sensé contrôler le procédé (un SCADA) aurait accès à toutes les variables.

— Assurer le respect de permissions (ex : lecture seule) sur des variables. Cette propriété de contrôle d'accès impose à un client de ne pouvoir faire des requêtes que de certains types (lecture, écriture, etc.) sur certaines variables, tout autre requête étant rejetée. Ces permissions peuvent d'une part être déclarées au niveau des clients (par exemple, un client avec peu de droits (un MES) ne pourrait effectuer que des requêtes de lecture afin de mesurer la production sans affecter le procédé). D'autre part, ces permissions peuvent être déclarées au niveau des variables elles-mêmes (par exemple une variable indiquant la valeur d'un capteur ne devrait pas être écrite par un client mais seulement lue). Enfin, des configurations plus fines peuvent être envisagées (un même client utilisé tantôt par un administrateur et tantôt par un opérateur devrait avoir des droits différents suivant qui le contrôle).

Propriétés orientées métier :

— N'autoriser qu'un certain ensemble de valeurs pour des variables numériques (ou booléennes). Cette propriété, liée à la sûreté de fonctionnement, impose à un client de ne pouvoir écrire dans une variable qu'une valeur autorisée (ici un nombre dans un intervalle donné). Elle est par exemple utile pour des variables dont les domaines de valeurs qu'elles devraient réellement pouvoir prendre est plus restreint que les valeurs possibles pour leurs types (par exemple un UInt8 compris entre 1 et 10). Cette propriété peut être généralisée à des types plus complexes comme des chaînes de caractères, des fichiers, etc.

— Borner le nombre de commandes passées durant un temps donné. Cette propriété, liée à la disponibilité « applicative », impose à un client de ne pas pouvoir envoyer plus d'un certain nombre de requêtes à un serveur pendant un temps donné. Elle peut par exemple être utile lorsqu'une commande sur un équipement demande un certain temps de traitement et que l'équipement ne doit pas être modifié entretemps (par exemple dans le cas du projet Aurora). Cette propriété peut être raffinée pour ne contrôler par exemple que les écritures (ou encore que les écritures d'une certaine valeur).

— Assurer des règles métier personnalisées pouvant dépendre de l'état observé du système. Ces règles intrinsèquement liées à la sûreté de fonctionnement visent à implémenter des filtres au moyen de constructions *Si-Alors-Sinon*. Ces règles permettent ainsi de vérifier des propriétés sur des suites de messages telles que « Si un disjoncteur est ouvert, alors on ne peut l'ouvrir avant de l'avoir fermé » ou

alors « Si la porte du four est ouverte, on ne peut pas le démarrer ». La sémantique de ces règles sera définie plus longuement en sections 3.2 et 3.3.

Dans la suite de ce chapitre, nous allons détailler les différents langages de configuration du filtre afin d'expliquer comment nous vérifions les propriétés décrites ci-dessus. Nous illustrons notre propos avec l'exemple de l'attaque Maroochy Shire introduit en sections 1.4 et 2.1.2. La figure 3.2 montre une cuve reliée à une pompe. La cuve se remplit au fil du temps et un capteur détecte lorsque le niveau de liquide est trop élevé. Lorsque ce niveau est atteint, la pompe se met automatiquement en marche. Il est possible de démarrer et d'arrêter la pompe en lui envoyant directement des commandes. On souhaite garantir que la pompe ne peut pas être arrêtée par le client tant que le niveau de liquide n'est pas redescendu sous la limite du capteur [1]. Une description plus détaillée de cet exemple et de différentes topologies réseau pouvant mener ou non à des attaques est également proposée en annexe A.4.

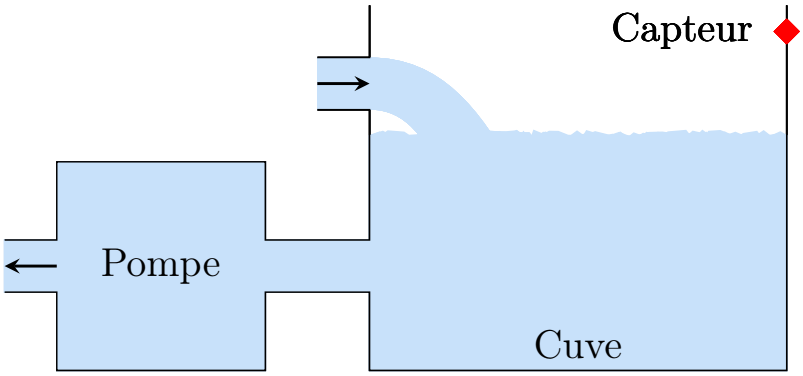


FIGURE 3.2. – Attaque Maroochy Shire

Nous adaptons cet exemple afin de motiver les fonctionnalités du filtre. On considère un unique serveur OPC-UA qui contrôle l'ensemble du procédé Son adresse IP est 10.0.0.5 et son port d'écoute étant 4840. Un unique client lit l'état du capteur et contrôle la pompe via des requêtes OPC-UA. Le serveur contrôle le procédé via les variables suivantes [2] :

— ns=5 ;s=Cuve.Niveau : Variable booléenne qui représente le niveau de la cuve grâce au capteur (True signifie que la cuve est pleine).
— ns=5 ;s=Pompe.Etat : Variable de type Int8 qui contrôle l'état de la pompe (-1 = marche arrière / 0 = arrêt / 1 = marche avant).

Les variables de la cuve sont décrites dans un fichier XML OPC-UA présenté en annexe B.1. Pour cet exemple, on souhaite garantir les propriétés de sûreté de fonctionnement suivantes :

1. A l'opposé, garantir que la pompe démarrera une fois que le capteur est plein est une propriété de vivacité. Ces propriétés sont de façon générale difficiles à vérifier par du filtrage qui travaille sur une trace finie d'exécution. Elles ne sont par ailleurs pas celles visées par le filtre que nous présentons ici qui cherche à bloquer des commandes malveillantes et non à détecter si le procédé atteint un état donné (par exemple « la pompe démarre »).

2. Pour rappel, dans les noms de variables OPC-UA, ns=5 ;s=Bla signifie la variable Bla dans le *namespace* 5.

R_1 : Seul le protocole OPC-UA est autorisé et ne peut effectuer que des commandes en lecture et en écriture.

R_2 : Le client et le serveur ne peuvent parler qu'entre eux.

R_3 : Le client peut lire toutes les variables mais n'écrire que sur ns=5 ;s=Pompe.Etat.

R_4 : Les valeurs écrites sur la variable ns=5 ;s=Pompe.Etat ne peuvent être que -1, 0 ou 1.

R_5 : Lorsque la variable ns=5 ;s=Pompe.Etat change de valeur, elle ne doit pas recevoir de nouvelle requête pendant une minute, le temps que la pompe prenne en compte le changement.

R_6 : Le client ne doit pas envoyer plus de 100 commandes par minute.

R_7 : La pompe ne doit pas être arrêtée si la cuve est pleine. Cela signifie que la variable ns=5 ;s=Pompe.Etat ne doit pas être mise à 0 si la variable ns=5 ;s=Cuve.Niveau vaut True.

3.1.4. Chaîne de configuration du filtre

Nous avons décrit en section 2.3.1 le fonctionnement général du dispositif de filtrage. Nous nous focalisons ici sur le filtre, sur les propriétés qu'il peut vérifier et sur les langages utilisés pour sa configuration. La figure 3.3 montre ainsi la chaîne de configuration du filtre.

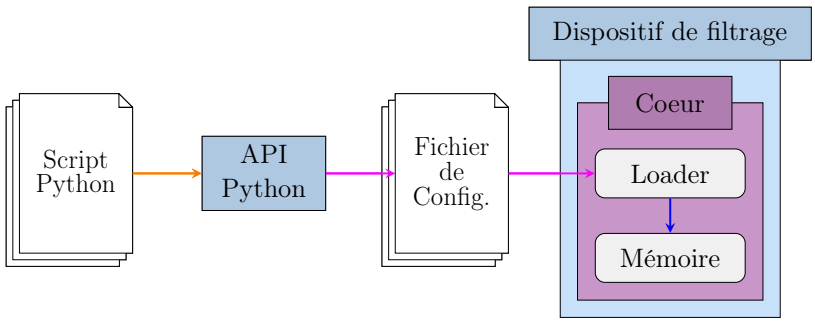


FIGURE 3.3. – Chaîne de configuration du filtre

Les requêtes des clients sont traduites dans une structure de données intermédiaire et commune à tous les protocoles implémentés sur le dispositif. Il est ainsi possible d'appliquer du filtrage sur tous les messages quelques soient leurs protocoles de communication. Pour cela, le filtre a à sa disposition une mémoire locale dans laquelle sont stockées les règles qu'il doit vérifier mais également des variables locales permettant de mémoriser des informations sur l'état du système. La configuration du dispositif et le chargement des règles sont effectués au moyen d'un langage textuel dit de « bas niveau » (représenté par une flèche rose →). Ce fichier est lu par un module de chargement, le *loader* qui stocke ces informations dans la mémoire afin qu'elles puissent être accédées par le filtre. Le loader assure que le fichier est lexicalement et syntaxiquement correct en s'appuyant sur la grammaire qui sera décrite en listing 3.2. Il vérifie également que la configuration lue est conforme aux contraintes de mémoire imposées. Pour faciliter et automatiser

partiellement la configuration du dispositif, une API permet de générer les fichiers bas niveau à partir de scripts Python (représenté par une flèche orange →).

On fait par ailleurs l'hypothèse que tous les messages entre un client et un serveur ne passent que par un seul dispositif. Dans le cas contraire, les dispositifs traversés devraient se synchroniser pour avoir une vision commune de l'état du système ce qui est compliqué dans le cas de réseaux industriels qui sont très figés (il serait compliqué d'introduire des communications entre les dispositifs). Cela introduirait par ailleurs de nouvelles problématiques de sécurité potentielles (par exemple si un attaquant supprime ou retarde les messages de synchronisation entre les dispositifs).

Problématique : Les problématiques inhérentes à tout mécanisme de filtrage d'une part l'expressivité des règles et la facilité de les exprimer. En effet, la puissance et la sécurité du filtre importent peu s'il n'est pas correctement configuré pour bloquer les attaques. Cette problématique est bien entendue vraie pour tout système d'information. Elle s'applique cependant particulièrement bien aux systèmes industriels du fait de leur faible variance dans le temps. Les systèmes industriels ont une topologie statique et un comportement n'évoluant peu ou pas. Leurs flux de données sont ainsi particulièrement adaptés aux vérifications à l'aide de règles de filtrage. Ainsi, pour répondre à ces problématiques, les contributions de cette thèse au projet ARAMIS, et plus particulièrement au filtre sont :

1. Une API Python orientée vers les systèmes industriels pour faciliter la spécification des configurations,
2. Des méthodes de génération automatiques de configuration vers le langage bas niveau à partir de l'API Python,
3. Des preuves de correction de la génération de code vers le langage bas niveau.

Nous avons aussi contribué à la définition initiale du langage bas niveau et de ses concepts clés (copies locales, isolation des canaux et utilisation de logique à trois valeurs) [Puys et al., 2016c], à l'algorithme de filtrage et à la définition de la forme intermédiaire des messages au sein du dispositif [Badrignans et al., 2017].

Plan du chapitre : La section 3.2 présentera le langage bas niveau et donne quelques résultats d'expressivité et de complexité de l'algorithme de filtrage. Ensuite, la section 3.3 décrira d'abord une API haut niveau et son utilisation en montrant comment concrètement configurer le filtre pour assurer la sécurité de l'exemple de l'attaque Maroochy Shire définie en section 3.1.3. Elle formalisera ensuite un processus de génération automatique des fichiers en langage bas niveau par l'API. Enfin la section 3.4 positionnera le filtre par rapport à l'état de l'art et discutera des possibilités d'extensions.

3.2. Langage bas niveau

On décrit sections 3.2.1, 3.2.2 et 3.2.3 le langage d'entrée du filtre, aussi appelé « langage bas niveau ». Ensuite, la section 3.2.4 discute des choix effectués. Enfin, la section 3.2.5 donne quelques résultats de complexité sur les opérations effectuées par le filtre et sur l'expressivité du langage bas niveau.

3.2.1. Objets manipulés dans le langage

Un fichier de configuration du filtre est constitué de deux parties, similairement aux deux catégories de propriétés que nous avons introduites en section 3.1.3. La première est la déclaration de la *topologie* du système industriel (clients, serveurs, variables, permissions, etc.). Viennent ensuite les règles orientées métier.

```
# Topologie
[Déclaration des serveurs]

[Déclaration des variables des serveurs]

[Déclaration des clients]

[Déclaration des canaux]

[Déclaration des droits d'accès et des permissions]

# Règles
[Déclaration des règles métier]
```

Listing 3.1 – Structure d'un fichier de configuration

Déclaration des serveurs : Les serveurs sont associés à un protocole de communication. Ils sont décrits par leurs informations réseau (adresse IP, port) en fonction du protocole, et se voient assignés un identifiant.

Déclaration des variables des serveurs : Sont ensuite définies les variables présentes sur les serveurs. Par défaut, toute commande portant sur une variable non déclarée sera par la suite rejetée.

Déclaration des clients : Les clients sont définis de manière analogue aux serveurs. En fonction du protocole de communication utilisé, ils peuvent également contenir des informations sur l'humain contrôlant le client.

Déclaration des canaux : Un serveur et un client sont liés par la notion de canal. Ce dernier sera crucial par la suite car il sera la cible de la quasi totalité des règles de filtrage.

Déclaration des droits d'accès et des permissions : Des permissions peuvent être affectées directement aux variables et aux canaux (par exemple afin de configurer les variables et/ou les canaux en lecture seule). Elles sont suivies par les droits d'accès des canaux aux variables (par exemple afin de limiter les variables auxquelles un canal peut accéder). Il est par ailleurs possible de spécifier des permissions plus fines sur les accès des canaux aux variables. Un exemple simple est un MES et un SCADA qui sont tous deux clients d'un automate programmable. Une variable critique pourra être écrite par le SCADA mais pas par le MES dont les permissions sont plus faibles. La variable sera accessible en écriture puisque le SCADA doit pouvoir y écrire. Le canal du MES pourra

potentiellement utiliser le service Write pour écrire sur des variables moins critiques. Mais son canal n'aura pas accès à la variable critique, réservée au SCADA.

Déclaration des règles métier : Enfin viennent les règles métier qui suivent le paradigme *Événement-Condition-Action* [Dittrich et al., 1995]. Dans notre cas, un événement est un accès à une variable par un canal via un ou plusieurs services donnés. Les couples Conditions/Actions sont spécifiés sous la forme de construction *Si-Alors-Sinon*. Les conditions et les actions peuvent faire appel à la mémoire courante du filtre. Cette mémoire est caractérisée par les valeurs de trois types d'objets : (i) les variables locales, (ii) les copies locales, (iii) les historiques.

Variable locales : Les variables locales sont internes au filtre et peuvent être lues et écrites dans les règles. Elles peuvent notamment servir à effectuer des calculs intermédiaires, à la manière des registres en assembleur. Cependant, leur valeur persiste entre le traitement des différents messages, permettant au filtre de garder un état interne.

Copies locales : Il est possible de garder des copies locales des variables lorsqu'elles sont impliquées dans des conditions de règles. Les variables à sauvegarder sont déclarées manuellement dans la partie règles métier du fichier de configuration. Alors, à chaque fois que la valeur de cette variable est précisée dans une commande, la copie locale est mise à jour en conséquence. Le filtre peut ainsi appliquer des règles en fonction de la valeur courante de ces variables. Évidemment, cette valeur comprend uniquement la vision du serveur par le filtre en fonction des messages échangés. Elle peut donc ne pas être celle réellement présente sur le serveur dans le cas où le procédé a évolué sans message. Cependant la vision d'un serveur qu'a le filtre est *à minima* celle du client. Ainsi si le filtre venait à prendre une mauvaise décision due à une valeur obsolète, l'opérateur sans le dispositif aurait également dû prendre une décision avec cette même valeur obsolète.

Historiques : Les historiques peuvent être vus comme des compteurs périodiques. Ils sont paramétrés par une limite de temps (période) et un seuil à atteindre. Plus formellement, un historique de période p et de seuil n est un tableau de n cases, contenant des horodatages : $h_{p,n} = \{t_1, .., t_n\}$. Ces compteurs sont incrémentés via un opérateur dans les règles du filtre et chaque incrément est horodaté. A la réception d'un message, les entrées de l'historique dépassant sa limite de temps sont supprimées. Ainsi à tout instant on a : $t_{max} - t_1 \leq p$ avec $max \leq n$. Si l'historique doit être incrémenté alors qu'il est déjà plein (contient n éléments), l'entrée la plus récente remplace la plus vieille (politique de cache LRU – *Least Recently Used*). Le compteur peut être incrémenté dans les règles du filtre et est donc décrémenté automatiquement avec le temps qui passe. Il est alors possible de tester le fait qu'il ait atteint son seuil dans la condition d'une règle. La figure 3.4 montre par exemple un historique de seuil 10 contenant actuellement 5 entrées.

3.2.2. Syntaxe des fichiers de configuration du filtre

Cette section présente la syntaxe des règles métier sous forme d'une grammaire en forme de Backus-Naur étendue, suivie d'un exemple en listing 3.3. Pour rappel, les non-

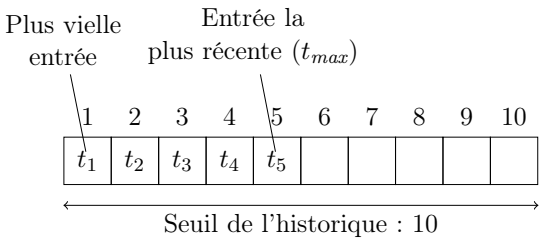


FIGURE 3.4. – Représentation d'un historique

terminaux sont entourés de chevrons <>, les terminaux par des guillemets "". Tous les autres caractères sont des opérateurs de la grammaire, à savoir :

— Les barres verticales | désignent un choix entre des éléments :
`<Booleen> ::= "True" | "False"`

— Les parenthèses () désignent un groupe d'éléments :
`<Operation> ::= <Entier> ("+" | "-" | "*" | "/") <Entier>`

— Les crochets [] désignent un groupe d'éléments optionnels :
`<Entier> ::= ["-"] <Nombre>`

— Les accolades {} désignent un groupe d'éléments pouvant se répéter :
`<Nombre> ::= <Chiffre> {<Chiffre>}`

Le listing 3.2 décrit ainsi la syntaxe des règles métier (notées <RegleMetier>), composées d'un événement (noté <Evenement>), et d'une suite de conditions (notées <Condition>) et d'actions (notées <Actions>).

```
<RegleMetier>    ::= "Filter" <Evenement> <ConditionsActions>

<Evenement>      ::= <ListeServices>  <Cible>
<ListeServices>  ::= <Service> {"," <Service>}
<Service>        ::= "Read" | "Write" | "Browse" | "Subscribe"

<Cible>          ::= <CanalSeul> | <VariableSeule> | <CanalVariable>
<CanalSeul>      ::= "Channel" <Id> ("Pre" | "Post")
<VariableSeule>  ::= "Variable" <Id> ("Pre" | "Post")
<CanalVariable>  ::= "Channel" <Id> "Variable" <Id>

<ConditionsActions> ::= <ConditionAction> {";" <ConditionAction>}
<ConditionAction>   ::= ["If" <Condition>] ["Then" <Actions>]
                        ["Else" <Actions>] ["Undef" <Actions>]

<Condition>      ::= <VarLocale> | <CopieLocale> | <HistPlein>
<Actions>        ::= <Action> {";" <Action>}
<Action>         ::= <Affectation> | <IncrementHist> | <Log> | <Verdict>

<Verdict>        ::= ("Reject" | "Accept") "(" <Str> ")"
<Log>            ::= "Log" "(" <Str> ")"
<Affectation>    ::= <VarLocale> ":=" <Expr>
<VarLocale>      ::= "Local(" <Id> ")"
<CopieLocale>    ::= "Copy(" <Id> ")"
<HistPlein>      ::= "IsFull(" <Id> ")"
<IncrementHist>  ::= "Touch(" <Id> ")"
```

```
<Expr>          ::= # Expressions arithmétiques et logiques
                    # sur des arguments de type <Arg>.

<Arg>           ::= "Value" | <Constante> | <VarLocale>
                  | <CopieLocale> | <HistPlein>

<Id>            ::= # Entier strictement positif
<Str>           ::= # Chaîne de caractères.
<Constante>     ::= # Nombre décimal.
```

Listing 3.2 – Grammaire des règles métier (en forme EBNF)

Exemple : Le listing 3.3 montre une règle assurant que la nouvelle valeur écrite par le canal 10 sur la variable 5 est comprise entre -1 et 1. Pour cela, les deux premières sous-règles calculent le fait que la nouvelle valeur de la variable est inférieure (resp. supérieure) à -1 (resp. 1) dans la variable locale 1 (resp. 2). Ensuite une troisième sous-règle calcule la conjonction des variables locales 1 et 2 dans la variable locale 3. Enfin, une dernière sous-règle teste la valeur de la variable locale 3 et rejette la commande si la variable locale 3 est évaluée à True (i.e. : si la nouvelle valeur est strictement inférieure à -1 ou supérieure à 1).

```
# Assure que la nouvelle valeur écrite par le canal 10
# sur la variable 5 est comprise entre -1 et 1.
Filter Write Variable 5 Channel 10          \
    Then Local(1) := Value < -1             \
    Then Local(2) := Value > 1             \
    Then Local(3) := Local(1) || Local(2) \
    If Local(3) Then Reject("Invalid new value for variable")
```

Listing 3.3 – Exemple de règle

Comme introduit en section 3.2.1, les règles métier sont exprimées par le formalisme *Événement-Condition-Action*. Un événement `<Evenement>` (Filter Write Variable 5 Channel 10 dans le listing 3.3) est un ensemble de services `<ListeServices>` et une cible `<Cible>`. La cible peut prendre trois formes :

— Un accès à n'importe quelle variable par un certain canal (noté `<CanalSeul>` dans la grammaire) : par exemple le canal 10 fait une écriture,

— Un accès à une certaine variable par n'importe quel canal (noté `<VariableSeule>` dans la grammaire) : par exemple la variable 5 est écrite,

— Un accès à une certaine variable par un certain canal (noté `<CanalVariable>` dans la grammaire) : par exemple le canal 10 écrit sur la variable 5. C'est le cas du listing 3.3.

Pour des contraintes d'implémentation et d'optimisation, il a été décidé que les règles portant sur chacune des trois formes de cibles décrites ci-dessus seront vérifiées indépendamment. Cela signifie que les règles sont classées en fonction de leur cible et que, par exemple, toutes les règles ayant pour cible un canal seul seront exécutées avant les règles portant un canal et une variable quelque soit leur ordre d'apparition. Cependant,

il apparaît que l'ordre dans lequel exécuter ces « catégories » de règles n'est pas trivial. Il y a par exemple des cas où l'on souhaite que les règles sur des canaux seuls soient exécutées avant les règles sur des canaux et des variables et vice versa. On propose d'introduire les mots-clés dans le cas des règles sur canaux seuls et variables seules afin que l'utilisateur puisse maîtriser l'ordre d'évaluation (mot-clé Pre signifiant « avant » et mot-clé Post « après »). Cela permet donc de répartir les règles en cinq catégories (énumérées ci-dessous). Dans chaque catégorie, l'ordre d'évaluation des règles est celui dans lequel elles apparaissent dans le fichier de configuration. L'ordre d'évaluation des catégories de règles est le suivant :

1. canal seul Pre ;
2. variable seule Pre ;
3. canal et variable ;
4. variable seule Post ;
5. canal seul Post.

La règle est ensuite divisée en sous-règles (notées `<ConditionAction>` dans la grammaire) qui seront exécutées séquentiellement. Chaque sous-règle est une structure If-Then-Else, le bloc If étant une condition `<Condition>` et les blocs Then et Else étant une suite d'actions `<Actions>`. Dans certains cas, les blocs If, Then et Else peuvent être optionnels. On peut par exemple avoir des sous-règles commençant directement par un bloc Then afin d'effectuer des calculs intermédiaires qui seront utilisés dans les conditions suivantes. De façon similaire, on peut avoir des sous-règles If cond Else action, correspondant à If Not cond Then action. *In fine*, les règles métier peuvent être vues comme des fonctions où les événements sont les paramètres et les sous-règles les instructions. On peut par exemple voir le listing 3.3 de la façon suivante :

```
def maRegle(variable, canal):
    temp1 = variable.value < -1
    temp2 = variable.value > 1
    temp3 = temp1 || temp2
    if temp3:
        return REJECT
    else:
        return ACCEPT
```

Les conditions sont des prédicats sur :

— le fait que des historiques soient pleins ou non (noté `<HistPlein>` dans la grammaire) ;

— les valeurs d'une copie locale ou d'une variable locale de type booléen (notées respectivement `<CopieLocale>` et `<VarLocale>` dans la grammaire).

Enfin, les actions peuvent être :

— l'affectation d'une expression `<Expr>` à une variable local `<VarLocale>` ;

— l'incrémentation d'un historique (notée `<IncrementHist>`) ;

— l'impression d'un événement de journalisation (notée `<Log>`) ;

— ou la prise d'une décision (accepter ou refuser la commande), On appelle cette décision un verdict (noté `<Verdict>`).

3.2.3. Algorithme du filtre

Pour rendre ses verdicts, le filtre a deux primitives : Accept et Reject. Pour vérifier une commande, le filtre vérifie les règles une par une dans l'ordre de leur apparition.

1. En cas de Accept, la commande est acceptée de façon pure et simple sans regarder les autres règles et sous règles. On pourrait le qualifier de « **force accept** ». Ce verdict est utile lorsqu'un cas particulier serait bloqué par des règles et qu'on souhaite le mettre en « liste blanche ».
2. En cas de Reject, la commande est rejetée de façon pure et simple sans regarder les autres règles.
3. En absence de verdict, le filtre passe à la sous-règle puis à la règle suivante. Enfin, en l'absence de verdict une fois toutes les règles évaluées, la commande est acceptée.

En fonction du protocole, il est possible qu'un message soit constitué d'une suite de commandes à vérifier. L'évaluation séquentielle des règles et sous-règles dans le cas d'une suite de commandes est présentée en algorithme 1.

Algorithme 1 Évaluation des messages par le filtre

Require: *REGLES* est l'ensemble des règles configurées sur le filtre.
Return: True si le message est accepté par le filtre, False sinon.

```
1: function EVALUERMESSAGE(m = [c_1, .., c_n])
2:     for all c ∈ m do
3:         for all r ∈ REGLES do
4:             verdict ← EVALUERRÈGLE(c, r)
5:             if verdict = REJECT then
6:                 return False
7:             else if verdict = ACCEPT then
8:                 break
9:     return True
```

On remarque donc qu'en cas de Accept, le filtre vérifie la commande suivante du message s'il y en a plusieurs. A l'opposé en cas de Reject, le filtre rejette le message dans son ensemble sans regarder les autres commandes s'il y en a plusieurs (et indépendamment du fait que l'une d'elles ait déjà été acceptée). Le rejet est donc prioritaire sur l'acceptation par application du principe de sûreté. Il serait néanmoins possible de modifier l'algorithme pour laisser le choix à l'utilisateur.

3.2.4. Discussion des choix effectués

On discute ici de quelques choix d'implémentation sur le dispositif de filtrage.

Services particuliers : Deux services sont mentionnés dans la grammaire et sont filtrés de façon différente des lectures et des écritures. Il s'agit des services mot-clé Browse et mot-clé Subscribe. mot-clé Browse permet à un client OPC-UA de connaître les fils d'un nœud (l'espace d'adressage du protocole OPC-UA étant un graphe). mot-clé Subscribe permet à un client de *s'abonner* à une variable. Le serveur enverra alors périodiquement

la valeur des variables auxquelles le client est abonné sans que celui-ci ne fasse de requête de lecture. Seules des propriétés basées sur la topologie réseau telles listées en section 3.1.3 (des propriétés de contrôle d'accès) sont considérées dans le cadre de ces deux services. Par exemple, il est possible d'interdire à un client d'explorer certains nœuds ou de s'abonner à certains nœuds.

Problématique de l'isolation des canaux : Les historiques, copies locales et les variables locales sont liés à un canal pour des raisons d'isolation. En effet, avoir un historique global à tous les canaux serait dangereux car un attaquant pourrait par exemple envoyer des commandes jusqu'à atteindre le niveau maximal autorisé par une règle sur l'historique. Le filtre bloquerait alors les commandes suivantes en accord avec la règle, y compris les commandes légitimes des opérateurs arrivant sur des canaux non corrompus. Des problématiques similaires pourraient se poser avec les copies locales. Il reste cependant possible de partager explicitement une copie locale ou un historique entre plusieurs canaux. Cela peut par exemple être utile lorsqu'un client fait toutes les lectures et un autre fait toutes les écritures.

Problématique de l'évaluation des conditions : il se peut dans certains cas que le filtre ne soit pas en mesure de prendre une décision. Cela peut arriver par exemple lorsqu'une règle est évaluée en fonction d'une copie locale dont on a jamais vu passer la valeur. Pour gérer ces situations, nous utilisons la logique à trois valeurs de Kleene [Kleene, 1952]. Cette logique introduit une valeur dite « inconnue » ou « inapplicable ». Les conditions peuvent donc être évaluées à la valeur Undef en plus de Then et Else. Dans le cas des opérations arithmétiques (notées `<Expr>` dans la grammaire), Undef est absorbant. Ainsi, afin de traiter ce cas il est possible de spécifier une suite d'action supplémentaires à celles du Then et du Else qui sera exécutée en cas de Undef.

Problématique de la mise à jour des copies locales pendant le filtrage : Si les demandes d'abonnement sont bien sûr filtrées (mot-clé Subscribe), pour des contraintes de performances il a été décidé que les notifications arrivant sur le dispositif de rupture ne passeraient pas par le filtre. Cependant, les copies locales sont tout de même mises à jour par ces notifications afin de maintenir un état le plus récent possible. Il est donc possible qu'une copie locale soit mise à jour pendant que le filtre applique ses règles (une notification arrivant au milieu du filtrage d'une requête). Une partie des règles sera donc vérifiée avec l'ancienne valeur de la copie locale et l'autre partie avec la nouvelle. Ce genre de comportement est source de *race conditions* et peut changer le comportement du filtre. Plusieurs solutions peuvent être proposées avec chacune des points positifs et négatifs :

1. Continuer de filtrer les autres commandes du message avec l'ancienne valeur (en bloquant la notification ou en stockant la nouvelle valeur en attendant la fin du filtrage). Cette solution a l'avantage de filtrer l'ensemble du message de façon cohérente mais délibérément avec une valeur obsolète de la copie locale.
2. Recommencer le filtrage avec la nouvelle valeur de la copie locale. Cela retarde le message et boucle à l'infini si jamais la fréquence de notification est plus rapide

que le temps de filtrage (peu probable dans les communications entre les niveaux 2 et 3 mais envisageable entre les niveaux 1 et 2).

3. Ne jamais tester directement la valeur d'une copie locale dans une règle mais faire au préalable une copie dans une variable locale. La copie locale est mise à jour en temps réel par la notification et la variable locale garde l'ancienne valeur. C'est une façon simple de simuler la première solution. Cependant, cette solution peut se révéler coûteuse si un nombre important de copies locales sont utilisées.

3.2.5. Quelques résultats de complexité et d'expressivité

Comme expliqué dans l'algorithme 1 en section 3.2.3, le filtre vérifie chaque règle de sa configuration pour chaque commande de chaque message arrivant. Les règles sont constituées d'une suite de conditions et d'actions. Afin d'assurer que le filtrage soit fait en temps et en mémoire bornée, les conditions et les actions doivent être appliquées en temps et coût mémoire constants. Dans notre cas, les conditions sont exclusivement des expressions arithmétiques et logiques, vérifiés sur des variables scalaires (pas de comparaison de chaînes de caractères par exemple). Elles sont donc vérifiées en temps et coût mémoire constants.

Par ailleurs, les actions sont limitées à : (i) bloquer ou laisser passer la commande, (ii) journaliser une information, (iii) mettre à jour une variable locale ou un historique, ces actions étant également effectuées en temps et coût mémoire constants. Ainsi, la vérification d'une commande dépend uniquement du nombre de règles. Dans le pire des cas (un message ne violant aucune règle), la vérification doit être effectuée sur toutes les règles. Donc, si l'on associe un temps (resp. coût mémoire) constant τ_i à chaque prédicat (condition et action) P_i apparaissant n_i fois au total dans le fichier de règles, on peut calculer le pire temps d'exécution (resp. coût mémoire) T d'une commande, tel que :

$$T = \sum \tau_i n_i$$

Les propriétés de sûreté sont définies comme des ensembles de traces valides. En particulier, elles ont la particularité que, une fois violée, elles ne peuvent plus être à nouveau vraies. Cependant, toutes les propriétés de sûreté ne sont pas exprimables au moyen d'automates d'états finis. En particulier, les propriétés ne pouvant être reconnues par un automate d'état finis ne peuvent être vérifiées en mémoire bornée (elles nécessitent par exemple un automate à pile). Afin de rester en mémoire bornée, le filtre se limite donc à des propriétés régulières portant sur les commandes envoyées.

3.3. API haut niveau

On décrit dans cette section une API générant des fichiers en langage bas niveau afin de faciliter la configuration du filtre. La section 3.3.1 décrit l'API en montrant comment rédiger un fichier de règles pour l'attaque Maroochy Shire. Ensuite, la section 3.3.2 discute de l'intérêt de l'API vis-à-vis du langage bas niveau. Enfin, la section 3.3.3 explique comment l'API génère automatiquement des fichiers en langage bas niveau.

3.3.1. Description de l'API Python

D'après une étude menée dans [Wool, 2004], environs 80% des pare-feux, tous milieux confondus, comportent des erreurs de configuration. Le dispositif de filtrage étant un pare-feu applicatif, il devra également faire face à ce problème. Ainsi, pour faciliter sa configuration, nous proposons une API (interface de programmation) Python qui permet de générer les fichiers de configuration du filtre en langage bas niveau. L'utilisation d'une API pour la configuration présente plusieurs avantages tirant partie des fonctionnalités des langages orientés objets (tel que Python dans notre cas).

Extensibilité : Dans un langage objet, l'utilisateur peut définir ses propres objets pour donner une sémantique à un ensemble de variables. Par héritage, si des règles de filtrage sont définies sur un objet, alors les objets le spécialisant seront également soumis à ces règles. Ces objets pourront remplacer ces règles ou en ajouter de nouvelles qui ne concernent qu'eux. Ainsi, il sera possible de définir l'objet Pompe qui aura des règles propres à une pompe (par exemple une limite sur la vitesse de rotation qu'on lui demande). Celui-ci pourra ensuite être spécialisé pour chaque marque ou modèle de pompe (par exemple PompeSchneider). Il sera alors possible de spécifier qu'une règle s'appliquant à toutes les pompes s'applique d'une certaine façon sur une pompe Schneider, puis d'ajouter des règles propres aux pompes de cette marque.

Composabilité : Grâce à la modularité des langages modernes, les règles de filtrage s'appliquant à chaque composant (par exemple la PompeSchneider) peuvent n'être définies qu'une seule fois et ensuite réutilisées au besoin à chaque fois que le composant en question apparaît dans un système. Il ne reste alors plus qu'à définir les règles à respecter pour les interactions des composants (par exemple la pompe ne doit pas être arrêtée tant que le capteur de la cuve détecte du liquide).

Simplicité d'utilisation : En concevant une API reposant sur un langage existant (ici Python) et non un langage ad hoc, on profite de toutes les fonctionnalités d'un langage de programmation moderne (boucles, fonctions, etc.). Qui plus est, la présence de variables dans le langage permet naturellement à l'utilisateur de raisonner sur des noms explicites, là où le langage bas niveau impose des identifiants numériques. Par exemple, il parait naturellement plus aisé de définir une règle sur la variable Pompe en fonction de la valeur courante de la variable Capteur de la cuve et non sur la variable 1501 dépendant de la copie locale 48.

Enfin, le choix de Python est motivé par l'existence d'un module libre python-opcua [3] disposant de représentations standards des données. Par ailleurs, ce module est en mesure de lire un fichier XML décrivant un espace d'adressage OPC-UA, moyen standard de représenter les objets dans le domaine. Cette fonctionnalité facilite et automatise ainsi grandement la configuration du dispositif en s'appuyant sur des fichiers représentant les données manipulées. En résumé, nous proposons une API qui permet de mémoriser les composants d'un système (lus dans un fichier XML via python-opcua et/ou entrés à la

3. https://github.com/FreeOpcUa/python-opcua

main). Elle rend ensuite possible la spécification de règles de filtrage et génère automatiquement un fichier en langage bas niveau correspondant. Cette API réalise également une série de vérifications en amont du chargement des règles dans le dispositif de filtrage, ceci afin de s'assurer que le fichier généré ne sera pas rejeté par le dispositif lors du chargement par le loader. Ces tests portent essentiellement sur la cohérence des informations de configuration. Ils vérifient par exemple que les permissions sur les éléments sont en adéquation les unes avec les autres (si une variable n'a pas les droits en écriture, alors aucun canal n'est sensé pouvoir écrire et aucune règle portant sur ses écritures ne doit apparaître). Pour générer un fichier de configuration du filtre, l'utilisateur a à sa disposition un ensemble de classes d'objets qu'il va instancier pour modéliser son système. Nous présentons ici les fonctionnalités principales de l'API en configurant le dispositif pour l'exemple de l'attaque Maroochy Shire présenté en section 3.1.3. Comme décrit plus haut, la structure d'un fichier de configuration est la suivante :

```
# Topologie
[Déclaration des serveurs]

[Déclaration des variables des serveurs]

[Déclaration des clients]

[Déclaration des canaux]

[Déclaration des droits d'accès et des permissions]

# Règles
[Déclaration des règles métier]
```

Listing 3.4 – Structure d'un fichier de configuration

3.3.1.1. Rappel de l'exemple de l'attaque Maroochy Shire

On rappelle rapidement en figure 3.5 l'exemple de l'attaque Maroochy Shire que l'on souhaite corriger en configurant le dispositif de filtrage.

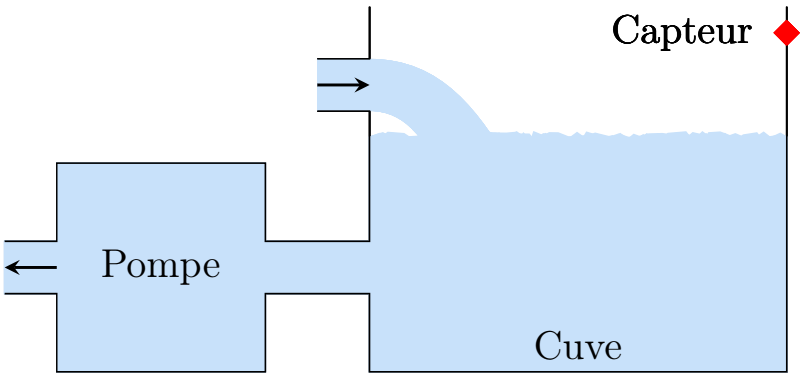


FIGURE 3.5. – Variation de l'attaque Maroochy Shire

Dans cet exemple, on considère qu'un unique serveur OPC-UA contrôle l'ensemble du

procédé, son adresse IP étant 10.0.0.5 et son port d'écoute étant 4840. Un unique client lit l'état du capteur et contrôle la pompe via des requêtes OPC-UA. Le serveur contrôle le procédé via les variables suivantes [4] :

— ns=5;s=Cuve.Niveau : Variable booléenne qui représente le niveau de la cuve grâce au capteur (True signifie que la cuve est pleine).

— ns=5;s=Pompe.Etat : Variable de type Int8 qui contrôle l'état de la pompe (-1 = marche arrière / 0 = arrêt / 1 = marche avant).

Les variables associées à la cuve sont décrites dans un fichier XML OPC-UA présenté en annexe B.1. Pour cet exemple, on souhaite garantir les propriétés de sûreté de fonctionnement suivantes :

R_1 : Seul le protocole OPC-UA est autorisé et ne peut effectuer que des commandes en lecture et en écriture.

R_2 : Le client et le serveur ne peuvent parler qu'entre eux.

R_3 : Le client peut lire toutes les variables mais n'écrire que sur ns=5;s=Pompe.Etat.

R_4 : Les valeurs écrites sur la variable ns=5;s=Pompe.Etat ne peuvent être que -1, 0 ou 1.

R_5 : Lorsque la variable ns=5;s=Pompe.Etat change de valeur, elle ne doit pas recevoir de nouvelle requête pendant une minute, le temps que la pompe prenne en compte le changement.

R_6 : Le client ne doit pas envoyer plus de 100 commandes par minute.

R_7 : La pompe ne doit pas être arrêtée si la cuve est pleine. Cela signifie que la variable ns=5;s=Pompe.Etat ne doit pas être mise à 0 si la variable ns=5;s=Cuve.Niveau vaut True. C'est la propriété de sûreté violée par l'attaque Maroochy Shire.

3.3.1.2. Description de la topologie du système industriel

L'utilisateur commence par déclarer le serveur OPC-UA du système.

```
srv = filtre.OpcUaServer("10.0.0.5", 4840)
```

Dans l'exemple, l'espace d'adressage de la cuve (i.e. : ses variables) est fourni dans le fichier XML, proposé en annexe B.1. Ce fichier peut être importé par l'objet représentant le serveur, créé ci-dessus, en lui indiquant le chemin du fichier XML de la façon suivante :

```
srv.importXml("cuve.xml")
```

Cependant dans notre exemple, seul l'espace d'adressage de la cuve est fourni dans un fichier XML. La variable contrôlant l'état de la pompe (ns=5;s=Pompe.Etat) est donc créée puis ajoutée à l'objet représentant le serveur. C'est par ailleurs le cas de toutes les variables MODBUS puisque ce protocole ne supporte pas une représentation en XML standardisée.

4. Pour rappel, dans les noms de variables OPC-UA, ns=5;s=Bla signifie la variable Bla dans le *namespace* 5.

```
# Crée une variable correspondant à l'état de la pompe.
etatPompe = filtre.OpcUaVariable("ns=5;s=Pompe.Etat", "Int8")
srv.addVariable(etatPompe)
```

L'utilisateur précise ensuite quels services sont autorisés sur les variables, de la même façon que dans le langage bas niveau (par exemple Read, Write, ou encore Browse et Subscribe dans le cas d'OPC-UA). Par défaut aucun service n'est autorisé donc tout service qui ne sera pas explicitement autorisé sera refusé. À noter que les services Browse et Subscribe ne peuvent pas être activés sur une variable MODBUS (puisque ce protocole n'implémente que les services Read et Write). Par ailleurs, il est possible de récupérer dans une liste toutes les variables ajoutées au serveur. Cette fonctionnalité peut par exemple servir à appliquer une même règle à toutes les variables.

```
# Autorise les services READ et BROWSE sur
# toutes les variables du serveur.
for loc in srv.getAllVariables().values():
    loc.enableRead()
    loc.enableBrowse()

# Autorise les écritures  sur la variable correspondant à l'état
# de la pompe afin qu'un client puisse la démarrer ou l'arrêter.
etatPompe.enableWrite()
```

Il est possible de rechercher une variable particulière sur le serveur afin de la réutiliser plus tard dans le script.

```
# Récupère la variable correspondant
# au capteur de niveau de la cuve.
capteur = srv.getVariableByNodeId("ns=5;s=Cuve.Niveau")
```

Les clients sont définis de façon analogue aux serveurs. Dans le cas d'OPC-UA, l'utilisateur peut alors spécifier un identifiant de session ("urn :MonApplicationURI" dans l'exemple). Ce dernier permet notamment d'identifier la personne utilisant le client car deux opérateurs sur le même client sont sensés avoir un identifiant de session différent. Il permet également de référencer un certificat comme moyen d'authentification grâce à l'ancre de confiance (le HSM dont est équipé le dispositif de filtrage comme expliqué en section 2.3.1).

```
# Configuration du client.
cli = filtre.OpcUaClient("10.0.1.5", "urn:MonApplicationURI")
```

Ensuite, un canal vient permettre au serveur et au client de communiquer.

```
# Configuration du canal.
chan = filtre.OpcUaChannel(srv, cli)
```

Les services autorisés sur le canal sont alors configurés de la même façon que les variables.

```
# Autorise les services READ, WRITE et BROWSE sur le canal.
chan.enableRead()
chan.enableWrite()
```

```
chan.enableBrowse()
```

De même, les services autorisés pour un canal sont définis pour chaque variable.

```
# Autorise le canal à accéder en READ et BROWSE
# sur toutes les variables du serveur.
for loc in srv.getAllVariables().values():
    chan.enableVariableRead(loc)
    chan.enableVariableBrowse(loc)

# Autorise le canal à écrire sur la variable
# correspondant à l'état de la pompe.
chan.enableVariableWrite(etatPompe)
```

Ainsi, la configuration du serveur, du client, du canal et des permissions ci-dessus décrit précisément les règles R_2 et R_3. De plus, cette configuration étant une liste blanche, toute commande impliquant un autre serveur, ou client sera refusée par le filtre.

3.3.1.3. Description des règles

Pour assurer la règle R_1, l'utilisateur va décrire des règles globales qui s'appliqueront à tous les canaux du dispositif de filtrage. Ces règles globales visent à autoriser ou interdire un service ou un protocole sur tout le dispositif. À nouveau, par sécurité, aucun service ni aucun protocole n'est activé par défaut. Ainsi, en l'absence de la première ligne toutes les requêtes de lecture seraient bloquées de même que tout trafic OPC-UA en l'absence de la dernière ligne.

```
# Règles globales.
filtre.GLOBAL_RULES.enableRead()
filtre.GLOBAL_RULES.enableWrite()
filtre.GLOBAL_RULES.enableBrowse()

# Autorise OPC-UA et fixe son port d'écoute à 4840.
filtre.GLOBAL_RULES.enableOpcUa(4840)
```

Pour la règle R_4, l'API Python propose une fonctionnalité permettant de limiter les plages de valeurs qu'une variable numérique peut prendre. Pour cela, il est possible de spécifier une valeur minimum et/ou maximum. Cette fonctionnalité sera ensuite automatiquement traduite en une règle en langage bas niveau. La variable contrôlant la pompe étant de type Int8 (entiers compris entre -128 et 127), lui donner une valeur minimum de -1 et maximum de 1 garantit qu'elle prendra uniquement les valeurs -1, 0 ou 1. Une variable de type flottant sera quant à elle limitée à toutes les valeurs étant dans l'intervalle autorisé. Cette fonctionnalité peut être généralisée à un ensemble explicite de valeurs.

```
# Limite les écritures possibles sur
# la pompe aux valeurs {-1, 0 ,1}.
etatPompe.setMinimumValue(chan, -1)
etatPompe.setMaximumValue(chan, 1)
```

Pour les règles R_5 et R_6, une autre fonctionnalité de l'API Python permet de limiter dans le temps le nombre de requêtes relatives à une variable. Seules les requêtes

d'écriture sont comptabilisées par défaut afin de ne pas perturber les logiciels des opérateurs qui font d'eux-mêmes périodiquement des lectures. Cependant, il est possible de comptabiliser les autres services en le précisant.

```
# Limite les changements d'état de la pompe à un par minute.
etatPompe.setSlowdown(chan, 60, 1)
```

En particulier dans le cas de la règle R_6, on souhaite limiter le nombre de commandes dans tout le réseau. Comme décrit en section 3.2.4, pour des questions d'isolation ces restrictions sont faites par canal et non globalement au dispositif.

```
# Limite toutes les requêtes sur le canal à 100 par minute.
chan.addSlowdown(
    60, 100, services=(
        filtre.Service.READ,
        filtre.Service.WRITE,
        filtre.Service.BROWSE
    )
)
```

Les règles sur les valeurs limites et sur la limitation du nombre de commandes dans le temps sont exprimées par des constructions complexes de règles en langage bas niveau. Afin de minimiser les erreurs de configuration du filtre, ces règles se présentent dans l'API comme des patrons de conception (*design patterns*). L'API assurant que la traduction de ces patterns sera correct par des preuves manuscrites présentées en section 3.3.3. Enfin, certaines règles intrinsèquement liées au procédé qui ne peuvent être exprimées par ces patterns, sont spécifiées à la main via l'objet Filter de l'API. C'est en particulier le cas de là règle R_7 : ne pas arrêter la pompe si le capteur de niveau a la valeur True. Elles sont alors exprimées dans le même formalisme qu'en langage bas niveau, c'est à dire via une suite de constructions *If-Then-Else*.

```
# La pompe ne doit pas être arrêtée
# si le niveau de liquide est trop haut :
rule = filtre.Filter(chan, etatPompe, filtre.Service.WRITE)
rule.addSubRule(
    condition=filtre.And(
        # Si le capteur détecte du liquide ...
        filtre.Equal(capteur.currentValue, 1),

        # Et que la commande qui arrive arrête la pompe ...
        filtre.Equal(filtre.NewValue(), 0)
    ),
    # Alors on rejète la commande.
    thenActions=filtre.Reject("La pompe ne doit pas être arrêtée !")
)
```

Enfin, à partir de ce script Python faisant des appels à l'API, l'utilisateur peut maintenant générer automatiquement le fichier de configuration du filtre correspondant en langage bas niveau. Pour l'exemple, le fichier obtenu est disponible en annexe B.2.

```
# Génère le fichier de configuration.
filtre.convert("cuve.cfg")
```

3.3.2. Qu'apporte l'API Python en plus du langage bas niveau ?

Pourquoi proposer un langage et une API alors que leurs fonctionnalités sont similaires ? Premièrement, en vue d'une certification Critères Commun EAL 6-7 du dispositif de filtrage, il ne paraissait pas judicieux d'embarquer un interpréteur Python (ou tout autre langage) sur le dispositif. Un langage ad hoc tel que le langage bas niveau nous oblige à développer notre propre interpréteur (qui est en l'occurrence le filtre). Ce faisant, nous risquons d'introduire des vulnérabilités dues à notre propre code. Cependant, cela nous permet de contrôler quelles fonctions du langage seront implémentées ou non et d'éviter toute exécution de code arbitraire (par exemple via la fonction os.system en Python). Deuxièmement, se restreindre à un langage *ad hoc* prive de l'utilisation des fonctionnalités intéressantes d'un langage de programmation. En effet, Python permet notamment de faire de boucles et des fonctions et propose de nombreux concepts de programmation orientée objet (héritage, polymorphisme, etc.). Ce point de vue d'informaticien peut ne pas être partagé par un automaticien. Cependant, tout le monde conviendra qu'il est plus simple de faire une boucle pour appliquer un même traitement à plusieurs dizaines de milliers de variables plutôt que de les traiter une par une. Le choix de Python est aussi basé sur le fait qu'il est enseigné depuis 2013 dans les classes préparatoires aux grandes écoles et certain lycées, formant les futurs opérateurs.

L'API apporte aussi plusieurs fonctionnalités pratiques à commencer par les patterns de règles décrits en section 3.3.3 qui automatisent l'expression de règles dans beaucoup de cas d'étude. Toujours dans l'idée de simplifier la rédaction des règles, les variables locales et les copies locales sont créées et manipulées directement par l'API (ce qui n'empêche pas l'utilisateur d'en créer lui-même). Enfin, plusieurs fonctionnalités « cosmétiques » ont été implémentées dans l'API. Ces fonctionnalités auraient pu être gérées dans le langage bas niveau mais au prix de complexifier le code du filtre. Il est par exemple possible de remplacer un verdict par un autre dans les règles du filtre de façon transparente. Cela peut être intéressant pour remplacer tous les Reject par des Log durant les phases de test sans avoir à modifier le fichier de configuration. Enfin, il est aussi possible d'ajouter un "niveau" à chaque règle sous la forme d'un entier. Cela permet d'activer ou de désactiver un ensemble de règles en fonction de leur niveau. Pour cela, un niveau minimum peut être défini dans les règles globales et toutes les règles ayant un niveau strictement supérieur à ce minimum seront ignorées (les niveaux les plus proches de zéro étant les plus importants de manière similaire aux *ring levels* dans les noyaux).

3.3.3. Génération de fichiers en langage bas niveau

Dans cette section, nous décrivons et formalisons les mécanismes de génération de fichiers en langage bas niveau à partir de scripts Python utilisant l'API. Nous présentons d'abord la traduction des objets simples comme les serveurs et les clients. Ensuite, nous abordons la gestion des arguments dans les conditions par l'API. Enfin nous formalisons les patterns de règles décrits en section 3.3.1 et leur traduction en langage bas niveau.

Traduction des objets simples : La traduction des objets simples est directe puisqu'un appel au constructeur de l'objet en Python se traduit par une ligne en langage bas niveau où les attributs de l'objet sont transposés en paramètres du langage bas niveau.

La traduction des serveurs est par exemple montrée en table 3.1. Le même principe s'applique aux clients, aux canaux et aux règles d'accès aux variables par les canaux.

Génération de langage bas niveau	

Python	`# Déclare un serveur OPC-UA et un serveur MODBUS.`
	`srvOpcUa  = OpcUaServer(ipOpcUa, portOpcUa)`
	`srvModbus = ModbusServer(ipModbus, portModbus, unitId)`

Bas niveau	`Server ID   Protocol Opcua  \`
	`    Addr ipOpcUa  Port portOpcUa`
	`Server ID+1 Protocol Modbus \`
	`    Addr ipModbus Port portModbus UnitId unitId`

TABLE 3.1. – Traduction de serveurs

Traduction des conditions : Comme décrit en section 3.2.2, la condition d'un Filter (notée <Condition> dans la grammaire en listing 3.2) doit être une variable locale, une copie locale ou le fait qu'un historique soit plein ou non. Ainsi, pour tester des conditions complexes, le résultat est calculé progressivement dans des variables locales à la manière de registres en assembleur. Afin de simplifier la tâche de l'utilisateur de l'API, il reste possible d'imbriquer des opérateurs entre eux dans les scripts Python. Lors de la génération du code en langage bas niveau, autant de variables locales que nécessaires sont créées et les différents calculs intermédiaires de la condition leur sont affectés. Par ailleurs les copies locales sont également masquées à l'utilisateur qui peut directement les référencer via un attribut currentValue, implicitement associé à toute variable. Les copies locales sont alors créées lors de la traduction et une variable dont l'attribut currentValue n'est pas appelé n'aura pas de copie locale. La nouvelle valeur d'une variable contenue dans la commande est quant à elle accessible via l'objet NewValue. La table 3.2 montre un exemple de condition basée sur l'amplitudevariation d'une variable par rapport à sa dernière valeur connue.

Génération de langage bas niveau

Python

```
# Rejette une commande sur la variable loc s'il
# provoque un changement de valeur d'amplitude
# supérieure à +/- M%.
rule = filtre.Filter(chan, loc, filtre.Service.WRITE)
rule.addSubRule(
    condition=filtre.GreaterThan(
        filtre.Absolute(
            filtre.Minus(filtre.NewValue(), loc.currentValue)
        ),
        filtre.Absolute(
            filtre.Div(
                filtre.Mult(loc.currentValue, M),
                100
            )
        )
    ),
    thenActions=filtre.Reject("Trop grande amplitude!")
)
```

Bas niveau	1 Filter Write Variable ID(loc) Channel ID(chan) \ 2 Then Local(1) := Value - Copy(1); \ 3 Local(2) := |Local(1)|; \ 4 Local(3) := Copy(1) * M; \ 5 Local(4) := Local(3) / 100; \ 6 Local(5) := |Local(4)| \ 7 Local(6) := Local(2) > Local(5) \ 8 If Local(5) Then Reject(\ 9 "Trop grande amplitude!" \ 10)

TABLE 3.2. – Traduction de conditions

Nous avons présenté en section 3.3.1 des patterns de règles tels que setMinimumValue ou setSlowdown. Ces patterns se traduisent par un enchaînement de plusieurs commandes en langage bas niveau. Leur traduction est ainsi automatisée afin de minimiser les erreurs de configuration du filtre. Plusieurs autres patterns sont proposés par l'API qui peut par ailleurs facilement être étendue.

Pattern 1 : Valeurs limites Ce pattern de règles permet de spécifier la valeur minimale et/ou maximale qu'une variable numérique scalaire peut accepter. Toute valeur strictement hors de l'intervalle sera alors rejetée. Les valeurs des limites entrées par l'utilisateur sont vérifiées avant traduction pour être cohérente avec le type de la variable (ex : la valeur maximum d'un UInt8 ne peut être de 2^{30} car cela dépasserait sa taille maximum sur machine $(2^8 - 1)$.) Ce pattern est utilisé en Python grâce à des méthodes associées aux variables (setMinimumValue et setMaximumValue) et sa traduction est présentée en table 3.3.

Génération de langage bas niveau	
Python	1 *# Limite les valeurs possibles de la* 2 *# variable loc à [Min;Max] sur le canal chan.* 3 loc.setMinimumValue(chan, Min) 4 loc.setMaximumValue(chan, Max) 5
Bas niveau	1 Filter Write Variable ID(loc) Channel ID(chan) \ 2 Then Local(X) := Value >= Min \ 3 Then Local(Y) := Value <= Max \ 4 Then Local(Z) := Local(X) && Local(Y) \ 5 If Local(Z) Else Reject(\ 6 "Invalid new value for variable ID(loc)" \ 7)

TABLE 3.3. – Traduction du pattern de valeurs limites

Théorème 1. *La traduction définie en table 3.3 garantit que l'écriture d'une nouvelle valeur* Value *pour une variable par un canal est acceptée si et seulement si* Min $\leq$ Value $\leq$ Max.

Démonstration. Lors d'une écriture d'une nouvelle valeur *Value* dans une variable par

un canal, trois cas sont possibles.

1. Si $Value < Min$, alors $Local(X) := False$ et donc $Local(Z) := False$. Le Reject ligne 5 sera exécuté et la commande sera refusée.
2. Si $Value > Max$, alors $Local(Y) := False$ et donc $Local(Z) := False$. Le Reject ligne 5 sera exécuté et la commande sera refusée.
3. Si $Min \leq Value \leq Max$, alors $Local(X) := True$ et $Local(Y) := True$ donc $Local(Z) := True$. Le Reject ligne 5 ne sera pas exécuté et la commande sera acceptée.

Dans les cas où seul le minimum (resp. maximum) est donné, alors seule $Local(X)$ (resp. $Local(Y)$) est testée dans la condition et la preuve est un sous-ensemble des cas 1 et 3 (resp. 2 et 3). □

Pattern 2 : Nombre maximum d'accès à une variable en une période de temps : Ce pattern permet de s'assurer qu'un canal n'accède pas plus qu'un certain nombre de fois à une variable durant une période de temps. Ce pattern est utilisé en Python grâce à une méthode associée aux variables (setSlowdown) et sa traduction est présentée en table 3.4. Le nombre d'accès à la variable est comptabilisé dans l'historique X. A chaque accès à la variable, les lignes 5 à 8 vérifie si l'historique est plein, indiquant que le nombre maximum d'accès a été atteint.

Génération de langage bas niveau	
Python	1 `# Limite les accès par le canal chan à la variable` 2 `# loc à M (exclu) en T secondes. Les accès sont` 3 `# comptabilisés si la condition cond est vérifié.` 4 `loc.setSlowdown(` 5 `chan, T, M, condition=cond, services=serv` 6 `)`
Bas niveau	1 `History X Channel ID(chan) Period T MaxAt M` 2 `Filter serv Variable ID(loc) Channel ID(chan) \` 3 `Then Local(Y) := cond \` 4 `If Local(Y) Then Touch(X) \` 5 `Then Local(Z) := IsFull(X) && Local(Y) \` 6 `If Local(Z) Then Reject( \` 7 `"Limit of access reached for variable ID(loc)" \` 8 `)`

TABLE 3.4. – Traduction du pattern de nombre d'accès à une variable

Théorème 2. *La traduction définie en table 3.4 garantit que sur une période de temps T, une variable ne sera pas accédée par un canal plus de $M - 1$ fois.*

Démonstration. Lors d'un accès à une variable par un canal, plusieurs cas sont possibles.

— Dans le cas où la condition *cond* est fausse, alors $Local(Y) := False$ et donc $Local(Z) := False$. Ni l'incrément ligne 4 ni le Reject ligne 6 ne seront faits.

— Dans le cas où la condition *cond* est vraie six cas restent possibles. Soit c_i une commande arrivant sur la variable par le canal lié à l'historique, c_1 la plus vieille

commande mémorisée dans l'historique, c_n la plus récente et t la fonction qui associe une commande à son horodatage. On a donc n commandes enregistrées dans l'historique. On veut montrer que la commande est acceptée si et seulement si $n < M$ après incrément ligne 4.

1. Si $t(c_i) - t(c_1) < T$ et $n < M - 1$, alors aucune commande ne sera supprimée de l'historique et l'incrément ligne 4 sera effectué. Une fois fait, on aura toujours $n < M$, ainsi $IsFull(X) = False$ et donc $Local(Z) := False$. Le Reject ligne 6 ne sera pas effectué et la commande sera acceptée.
2. Si $t(c_i) - t(c_1) < T$ et $n = M - 1$, alors aucune commande ne sera supprimée de l'historique et l'incrément ligne 4 sera effectué, rendant l'historique plein. Une fois fait, on aura $n = M$, ainsi $IsFull(X) = True$ et donc $Local(Z) := True$. Le Reject ligne 6 sera effectué et la commande est sera refusée.
3. Si $t(c_i) - t(c_1) < T$ et $n = M$, alors aucune commande ne sera supprimée de l'historique et l'incrément ligne 4 ne sera pas effectué, l'historique étant déjà plein. On aura toujours $n = M$, ainsi $IsFull(X) = True$ et donc $Local(Z) := True$. Le Reject ligne 6 sera effectué et la commande est sera refusée.
4. Si $t(c_i) - t(c_1) \geq T$ et $n < M - 1$, alors $k > 0$ commandes seront supprimés de l'historique, l'incrément ligne 4 sera effectué et on aura $n < M - 1$, ainsi $IsFull(X) = False$ et donc $Local(Z) := False$. Le Reject ligne 6 ne sera pas effectué et la commande sera acceptée.
5. Si $t(c_i) - t(c_1) \geq T$ et $n = M - 1$, alors $k > 0$ commandes seront supprimés de l'historique, l'incrément ligne 4 sera effectué et on aura $n \leq M - 1$, ainsi $IsFull(X) = False$ et donc $Local(Z) := False$. Le Reject ligne 6 ne sera pas effectué et la commande sera acceptée.
6. Si $t(c_i) - t(c_1) \geq T$ et $n = M$, alors $k > 0$ commandes seront supprimés de l'historique, l'incrément ligne 4 sera effectué et on aura $n = M - k + 1$, donc :
   - Si $k = 1$ on aura toujours $n = M$, ainsi $IsFull(X) = True$ et donc $Local(Z) := True$. Le Reject ligne 6 sera effectué et la commande est sera refusée.
   - Si $k > 1$, on aura $n < M$, ainsi $IsFull(X) = False$ et donc $Local(Z) := False$. Le Reject ligne 6 ne sera pas effectué et la commande sera acceptée.

Dans le cas où la condition n'est pas spécifiée, l'incrément ligne 4 sera toujours effectué on se reportera au cas correspondant parmi les six. □

Le pattern présenté en table 3.4 peut aussi s'appliquer pour un canal tout entier. Dans ce cas seul l'événement lié à la règle change pour s'activer à toutes les requêtes sur le canal et non plus seulement à une variable en particulier. L'événement de type `<CanalVariable>` devient donc un événement `<CanalSeul>`. Les sous-règles restent identiques. Ce pattern est utilisé en Python grâce à une méthode associée aux canaux (addSlowdown) et sa traduction est présentée en table 3.5.

Génération de langage bas niveau	

Python	`# Limite les accès par le canal chan à n'importe quelle`
	`# variable à M (exclu) en T secondes. Les accès sont`
	`# comptabilisés si la condition cond est vérifié.`
	`chan.addSlowdown(`
	`  T, M, condition=cond, services=serv, priority=Pre/Post`
	`)`

Bas niveau	`History X Channel ID(chan) Period T MaxAt M`
	`Filter serv Channel ID(chan) Pre/Post \`
	`  Then Local(Y) := cond \`
	`  If Local(Y) Then Touch(X) \`
	`  Then Local(Z) := IsFull(X) && Local(Y) \`
	`  If Local(Z) Then Reject( \`
	`  "Limit of access reached for channel ID(chan)" \`
	`  )`

TABLE 3.5. – Traduction du pattern de nombre d'accès pour un canal

Théorème 3. *La traduction définie en table 3.5 garantit que sur une période de temps T, un canal ne fera pas plus de M-1 requêtes.*

Démonstration. La preuve est identique à celle du théorème 2. □

Pattern 3 : Nombre maximum de battements de variables sur une période de temps

Ce pattern est similaire au pattern précédent à la différence près qu'il s'intéresse à la variation simultanée de plusieurs variables données. Ainsi chaque variable peut dépasser une limite de nombre d'accès choisie sans conséquence. Le filtre bloquera les communications sur le canal lorsque qu'un certain nombre de ces variables auront dépassé la limite de nombre d'accès. Plus formellement cette règle s'assure que sur un canal, il n'y aura pas plus de N variables d'un ensemble déterminé dont la valeur bat plus vite que toutes les P secondes sur une période d'observation de T secondes. Ce pattern est utilisé en Python grâce à une méthode associée aux canaux (addFlutteringSlowdown) et sa traduction est présentée en table 3.6.

Génération de langage bas niveau	

Python	`# Assure qu'au plus N des K variables parmi l'ensemble`
	`# [v1,...,vK] pourront battre plus rapidement que`
	`# toutes les P secondes sur une période d'observation`
	`# de T secondes.`
	`chan.addFlutteringSlowdown(`
	`  T, N, P, variables=[v1,...,vK], priority=Pre/Post`
	`)`

Bas niveau

```
1  History X Channel ID(chan) Period T MaxAt N
2  History Y1 Channel ID(chan) Period P MaxAt 2
3                  [...]
4  History YK Channel ID(chan) Period P MaxAt 2
5
6  Copy M1 Channel 10 Variable 1
7                  [...]
8  Copy MK Channel 10 Variable 3
9
10 Filter Write Variable ID(v1) Channel ID(chan)       \
11     Then Local(A1) := Value != Copy(M1)             \
12     If Local(A1) Then Touch(Y1)                     \
13     If IsFull(Y1) Else Local(B1) := False           \
14     Then Local(C1) := IsFull(Y1) && !Local(B1)      \
15     If Local(C1) Then Touch(X); Local(B1) := True
16
17                 [...]
18
19 Filter Write Variable ID(vK) Channel ID(chan)       \
20     Then Local(AK) := Value != Copy(MK)             \
21     If Local(AK) Then Touch(YK)                     \
22     If IsFull(YK) Else Local(BK) := False           \
23     Then Local(CK) := IsFull(YK) && !Local(BK)      \
24     If Local(CK) Then Touch(X); Local(BK) := True
25
26 Filter Write Channel ID(chan) Pre/Post             \
27     If IsFull(X) Then Reject(                       \
28         "Fluttering exceeded for channel: ID(chan)" \
29     )
```

TABLE 3.6. – Traduction du pattern de nombre de battements de variables pour un canal

Lemme 1. *La traduction définie lignes 10 à 15 et 19 à 24 de la table 3.6 détecte qu'une variable bat plus rapidement que toutes les P secondes et sinon incrémente une seule fois l'historique X.*

Démonstration. Lors d'un accès à la variable v_1 par un canal, la dernière valeur connue de v_1 est gardée dans la copie locale M_1. On veut montrer que l'historique X est incrémenté si et seulement si l'historique Y_1 est plein et que $Local(B_1) = False$ (dans le cas contraire on a déjà incrémenté X). Soit c_i une commande arrivant sur la variable v_1 par le canal lié à l'historique Y_1, c_1 le plus vieux message contenu dans l'historique, c_n le message le plus récent et t la fonction qui associe une commande à son horodatage. Dû à la ligne 15, avoir $Local(B_1) = True$ à l'arrivée d'une commande est équivalent à avoir $n = 2$. Plusieurs cas sont alors possibles.

— Si la nouvelle valeur *Value* de v_1 est la même que celle de la copie locale correspondante à M_1 alors la variable ne change pas de valeur (donc ne bat pas) et

l'incrément ligne 12 ne sera pas effectué [5]. La ligne 13 supprimera potentiellement de l'historique Y_1 les messages plus vieux que P secondes. Si $n < 2$ en sortie, on aura $Local(B_1) := False$, sinon on avait $Local(B_1) = True$ par hypothèse et la ligne 13 la maintiendra.

— Si la nouvelle valeur est différente alors cinq cas restent possibles.

1. Si $n = 0$ et $Local(B_1) = False$, alors aucun accès à la variable n'a été fait dans les P dernières secondes. Aucune commande ne sera supprimée de l'historique Y_1 (il est vide), l'incrément ligne 12 sera effectué et on aura $n = 1$. $Local(B_1)$ restera à *False* dû à la ligne 13. Un changement de valeur a donc bien été détecté mais la variation entre les deux valeurs a été faite dans un temps supérieur à P puisque l'historique Y_1 était vide. Ainsi $IsFull(Y_1) = False$ et donc $Local(C_1) := False$ et l'incrément de l'historique X ligne 15 ne sera pas effectué.
2. Si $n = 1$ et $t(c_i) - t(c_1) < P$ et $Local(B_1) = False$, alors un changement de valeur a déjà été détecté dans les P dernières secondes et la commande arrivant c_n constitue un battement. Aucune commande ne sera supprimée de l'historique Y_1, l'incrément ligne 12 sera effectué et on aura $n = 2$. $Local(B_1)$ restera à *False* dû à la ligne 13. Ainsi $IsFull(Y_1) = True$ et $Local(B_1) := False$ donc $Local(C_1) := True$. L'incrément de l'historique X ligne 15 sera effectué passant $Local(B_1)$ à *True* par la même occasion.
3. Si $n = 1$ et $t(c_i) - t(c_1) \geq P$ et $Local(B_1) = False$, alors un changement de valeur a déjà été détecté il y a plus de P secondes et la commande arrivant c_n constitue un battement mais d'une période supérieur à P. Au moment d'incrémenter la valeur de l'historique Y_1 à la ligne 12, le seul message qu'il conservait sera supprimé. Ainsi, on aura $n = 1-1+1 = 1$ et $IsFull(Y_1) = False$ et $Local(B_1) := False$ dû à la ligne 13 (quand bien même sa valeur était déjà *False*). Par ailleurs, $IsFull(Y_1) = False$ et donc $Local(C_1) := False$ et l'incrément de l'historique X ligne 15 ne sera pas effectué.
4. Si $n = 2$ et $t(c_i) - t(c_1) < P$ et $Local(B_1) = True$, alors un battement a déjà été détecté dans les P dernières secondes et l'incrément de l'historique X a déjà été effectué. Au moment d'incrémenter la valeur de l'historique Y_1 à la ligne 12 aucune commande ne sera supprimée et l'incrément ne sera pas effectué (l'historique étant déjà plein). Ainsi on aura donc toujours $n = 2$ et $Local(B_1)$ restera à *True* dû à la ligne 13. Par ailleurs, $Local(B_1) = True$ et donc $Local(C_1) := False$ et l'incrément de l'historique X ligne 15 ne sera pas effectué à nouveau.
5. Si $n = 2$ et $t(c_i) - t(c_1) \geq P$ et $Local(B_1) = True$, alors un battement a déjà été détecté il y a plus de P secondes et l'incrément de l'historique X a

5. On peut se demander pourquoi l'horodatage n'est tout de même pas mis à jour. Par exemple dans le cas où une commande mettant la variable à *True* arrive à un instant $t(c_1)$ puis une commande la mettant à nouveau à *True* arrive à l'instant $t(c_2) = t(c_1) + P + u, u > 0$ et enfin un dernier message mettant la variable à *False* à l'instant $t(c_3) = t(c_2) + v, v < P$. Certes la variable a reçu une commande de passage à *True* suivi d'un passage à *False* en un temps inférieur à P mais nous avons choisi de considérer que le passage de *True* à *True* du message c_2 sera vraisemblablement ignoré par le serveur (voir même bloqué par une autre règle) et que la référence à considérer dans ce cas est bien c_1.

déjà été effectué. Au moment d'incrémenter la valeur de l'historique Y_1 à la ligne 12, $1 \leq k \leq 2$ messages seront supprimés et l'incrément sera effectué. On aura donc $n = 2 - k + 1$ donc :

— Si $k = 1$ on aura $n = 2$, donc $Local(B_1)$ restera à *True* dû à la ligne 13 et $Local(B_1) = True$ et donc $Local(C_1) := False$ donc l'incrément de l'historique X ligne 15 ne sera pas effectué à nouveau.

— Si $k = 2$ on aura $n = 1$, donc $Local(B_1)$ repassera à *False* dû à la ligne 13 et $IsFull(Y_1) := False$ et donc $Local(C_1) := False$ donc l'incrément de l'historique X ligne 15 ne sera pas effectué.

La preuve est identique pour les variables de 2 à K avec leurs copies locales, historiques et variables locales respectives. □

Théorème 4. *La traduction définie en table 3.6 garantit que sur une période de temps T, au plus N variables battront plus rapidement que toutes les P secondes dû à des accès d'un même canal.*

Démonstration. Par application du lemme 1 pour chaque variable $[v_1,...,v_K]$, l'historique X contiendra exactement le nombre de variables qui battent plus rapidement que P secondes. Ainsi, en le définissant avec une période de T et un seuil de N, il détectera quand N variables battent plus rapidement que P secondes sur une période d'observation de T secondes. □

Pattern 4 : Nombre maximum de variations de variables sur une période de temps
Ce pattern a le même comportement que le précédent à la différence qu'il ne compte que les modifications sur les variables scalaires numériques qui ont une amplitude dépassant une limite par rapport à la valeur précédente. Plus formellement cette règle s'assure que sur un canal, il n'y aura pas plus de N variables d'un ensemble déterminé dont la valeur change plus vite que toutes les P secondes avec une amplitude de plus ou moins $M\%$ sur une période d'observation de Q secondes. Ce pattern est utilisé grâce à une méthode associée aux canaux (addVariationSlowdown) et sa traduction est présentée en table 3.7.

Génération de langage bas niveau	
Python	1 `# Assure qu'au plus N des K variables parmi l'ensemble` 2 `# [v1,...,vK] pourront changer de valeur plus rapidement` 3 `# que toutes les P secondes sur une période d'observation` 4 `# de T secondes avec une amplitude de +/-M%.` 5 `chan.addVariationSlowdown(` 6 `T, N, P, M, variables=[v1,...,vK], priority=Pre/Post` 7 `)`

Bas niveau

```
History X Channel ID(chan) Period T MaxAt N
History Y1 Channel ID(chan) Period P MaxAt 2
                [...]
History YK Channel ID(chan) Period P MaxAt 2

Copy M1 Channel 10 Variable 1
                [...]
Copy MK Channel 10 Variable 3

Filter Write Variable ID(v1) Channel ID(chan)       \
    Then Local(A1) := Value - Copy(M1);             \
         Local(B1) := |Local(A1)|;                  \
         Local(C1) := Copy(M1) * M;                 \
         Local(D1) := Local(C1) / 100;              \
         Local(E1) := |Local(D1)|                   \
         Local(F1) := Local(C1) > Local(E1);        \
    If Local(G1) Then Touch(Y1)                     \
    If IsFull(Y1) Else Local(H1) := False           \
    Then Local(H1) := IsFull(Y1) && !Local(H1)      \
    If Local(I1) Then Touch(X); Local(H1) := True

                [...]

Filter Write Variable ID(vK) Channel ID(chan)       \
    Then Local(AK) := Value - Copy(MK);             \
         Local(BK) := |Local(AK)|;                  \
         Local(CK) := Copy(MK) * M;                 \
         Local(DK) := Local(CK) / 100;              \
         Local(EK) := |Local(DK)|                   \
         Local(FK) := Local(CK) > Local(EK);        \
    If Local(GK) Then Touch(YK)                     \
    If IsFull(YK) Else Local(HK) := False           \
    Then Local(HK) := IsFull(YK) && !Local(HK)      \
    If Local(IK) Then Touch(X); Local(HK) := True

Filter Write Channel ID(chan) Pre/Post              \
    If IsFull(X) Then Reject(                       \
        "Variations exceeded for channel: ID(chan)" \
    )
```

TABLE 3.7. – Traduction du pattern de nombre de battements de variables pour un canal

Théorème 5. *La traduction définie en table 3.7 garantit que sur une période de temps T, au plus N variables changeront de valeur plus rapidement que toutes les P secondes avec une amplitude de plus ou mois $M\%$ dû à des accès d'un même canal.*

Démonstration. La preuve est identique à celle du théorème 4. La seule différence réside dans la condition en accord avec laquelle les messages sont comptabilisés. Là où la traduction définie en table 3.6 ne requiert que la différence de la nouvelle valeur avec la dernière valeur connue dans la copie locale, celle de la table 3.7 effectue un calcul plus complexe pour vérifier l'amplitude de la variation. En particulier, la condition vérifiée

ici est :

$$|Value - Copy(M_K)| > \left|\frac{Copy(M_K) \times M}{100}\right|$$

□

3.4. Conclusion

Dans ce chapitre, nous avons présenté un filtre applicatif dédié aux systèmes industriels et avons montré comment le configurer. Les règles prennent aussi en compte des propriétés orientées métier sur les valeurs possibles des variables, sur la temporisation des commandes et sur l'état global du système [Puys et al., 2016c, Badrignans et al., 2017]. Nous avons proposé des patterns de règles dans l'API permettant de générer des règles en langage bas niveau. De façon plus générale, nous avons montré dans ce chapitre comment un filtre applicatif (ici le filtre du dispositif de filtrage) peut aider à endiguer une variante de l'attaque Maroochy Shire. Pour conclure ce chapitre, cette section décrit comment notre filtre se place par rapport à l'état de l'art et quelles pourraient être les possibilités d'amélioration.

3.4.1. Positionnement par rapport à l'état de l'art

Notre filtre, au même titre que celui proposé par Chen et Abdelwahed [Chen and Abdelwahed, 2014], se base sur chaque message reçu pour décider si ce message doit être accepté ou non. A l'opposé, les travaux de Cox [Cox, 2011] et Cárdenas *et al.* [Cárdenas et al., 2011] se basent sur des relevés de l'état courant du système. Cela signifie que, dans leur cas, un message ne respectant pas la politique de sécurité ne sera détecté qu'une fois ses effets sur le système détectés. Par ailleurs, les travaux ci-dessus utilisent des méthodes d'apprentissage pour configurer le filtre et ainsi repérer si le comportement actuellement observé dévie du comportement de référence. Ces techniques sont extrêmement pratiques du fait qu'elles sont automatiques. Cependant, si l'état du système sur lequel est basé l'apprentissage est déjà sous attaque, alors la détection sera faussée. En comparaison, dans notre approche la configuration du filtre est écrite manuellement par un opérateur. Bien que cette tâche soit plus laborieuse et laisse la place à des erreurs possibles, elle assure que la configuration est claire et totalement connue des opérateurs. Enfin, comparer un état à un autre pour repérer des déviations se fait généralement via des fonctions de corrélation qui vont calculer l'intensité du lien entre ces états et émettre une alarme à partir d'un certain seuil. Notre filtre, lui, se base sur une suite de règles sous forme de prédicats qui n'introduisent aucune approximation. Les historiques proposés par notre filtre diffèrent des *shallow history automata* de Fong [Fong, 2004]. En effet, les *shallow history automata* ne sont ni ordonnés, ni remis progressivement à zéro dans le temps afin de gagner de la place en mémoire.

3.4.2. Problématique des commandes multiples

Nous avons évoqué en section 3.2.3 que dans le cas des protocoles MODBUS et OPC-UA, un message peut être composé d'une suite de commandes et l'ordre dans lequel ces commandes sont filtrées peut changer le verdict du filtre. Lire ou écrire plusieurs

variables MODBUS se fait en précisant le premier indice suivi du nombre de variable à traiter (on traite donc des plages contiguës de variables). Dans le cas d'OPC-UA, à notre connaissance, aucun ordre de variables n'est imposé au client lorsqu'il effectue une requête. Il peut donc très bien demander à écrire la variable X puis la variable Y ou l'inverse. Pour pallier ce problème, il a été décidé à la suite de discussion avec les membres du projet ARAMIS que toutes les commandes d'un message devraient être évaluées sur l'état du dispositif au moment de la réception dudit message. Ainsi, selon la classification de Chang *et al.* [Chang et al., 1993], le filtre entre clairement dans la catégorie sûreté du fait que toutes les décisions sont prises sur l'état actuel et passé du système.

On peut alors penser le filtrage comme un mécanisme de transaction. Cependant, parler de transaction implique un mécanisme d'annulation (*rollback*) en cas de problème avant la fin. Pour cela les copies locales ne sont mises à jour qu'une fois toutes les commandes acceptées. De la même façon, les variables locales créées par l'API ont une durée de vie liée à celle de chaque commande à filtrer afin d'empêcher que ces variables interfèrent avec le filtrage des commandes suivantes. Enfin dans le cas des historiques, la question du *rollback* est discutable. En effet, l'absence de ce mécanisme induit une décorrélation entre le filtre et le serveur (le filtre incrémente l'historique alors que le serveur n'a pas reçu la commande qui a été bloquée). Cependant, une tentative d'accès a tout de même été faite, bien que rejetée à cause d'un autre message de la transaction. Suivant le contexte, on pourrait tout de même vouloir garder cette tentative en mémoire. Tels que sont actuellement les historiques, leur implémentation sur le dispositif ne prévoit pas de mécanisme de *rollback* afin de remettre l'historique à son état au moment avant réception.

Enfin, l'idée de filtrer tous les messages d'une transaction sur l'état au moment de la réception est plausible et est intrinsèquement liée à des exigences du projet mais elle peut néanmoins être discutée et on peut imaginer au moins deux variantes.

Variante 1 : En restant dans la catégorie sûreté de la classification de Chang *et al.*, on pourrait par exemple envisager que l'état soit mis à jour au fur et à mesure et que le *i-ème* message de la transaction soit vérifié en fonction de l'état laissé par le message $i-1$. Un mécanisme de retour à l'état de la réception serait toujours nécessaire et le filtre deviendrait dépendant de l'ordre des commandes qui devrait être normalisé. Il serait par contre plus proche de l'état réel du système.

Variante 2 : En passant dans la catégorie *response* de Chang *et al.*, le filtre pourrait filtrer le message i en fonction de l'état courant et passé mais aussi en fonction de l'état du système après application des messages suivants dans la transaction. Cela pourrait être réalisé en appliquant une première passe de filtrage pour simuler l'état du système après tous les messages de la transaction puis en repassant sur chacun pour s'assurer que ces conditions sont validées.

Pour illustrer l'intérêt de ces variantes, on peut reprendre l'exemple de la cuve et de la pompe avec la propriété "ne pas arrêter la pompe si le niveau de liquide est trop haut". Si l'on reçoit une transaction avec les commandes ("Arrêter la pompe", "Remplir la cuve"), le filtre laissera passer le message puisque la cuve est vide au moment où le message

"Arrêter la pompe" est filtré et ce, quelque soit l'ordre puisque la copie locale sur le capteur de niveau ne sera mise à jour qu'une fois les messages acceptés. La variante 1 proposée ci-dessus bloquerait le message si l'ordre des commandes est "Remplir la cuve" puis "Arrêter la pompe". La variante 2, quand à elle, bloquerait le message quelque soit l'ordre des commandes. Les catégories *guarantee* et *persistance* de Chang *et al.* implémentent des propriétés de vivacité disant "un jour j'aurai cet événement" (par exemple la pompe démarre si la cuve est pleine). Pour les satisfaire, un message peut être retardé (en principe sur un temps borné) en attendant que l'événement requis se produise. Cependant les systèmes industriels étant avant tout des systèmes temps réel, il parait malaisé de retarder les commandes envoyées.

3.4.3. Généralisation du filtre et perspectives

Nous avons vu dans la section précédente plusieurs axes d'amélioration pour la prise en compte des commandes multiples. Un autre axe d'amélioration serait l'implémentation du filtrage sur les appels de méthodes à distance – RPC – (notamment le service Call dans le cas d'OPC-UA). Nous avons volontairement exclu ce type de service de part la complexité à gérer le risque inhérent à l'appel de code à distance. Cependant il pourrait être requis dans certaines installations et une réflexion sur sa prise en compte peut s'avérer judicieuse. Il serait aisé de vérifier les droits d'accès des canaux de la même façon que pour les services Read, Write et Browse. Enfin, l'appel d'une méthode, ainsi que la réponse du serveur, pourraient servir d'événements pour le déclenchement de règles. Cependant, autoriser des appels de méthodes pose une question non triviale : est-ce que le contenu de la méthode est en conflit avec des règles métier ? Par exemple, si un méthode incrémente une variable et que cette variable ne doit pas dépasser une certaine limite, comment savoir si un appel à la méthode peut ou non violer la règle. Il faudrait pour cela pouvoir émuler la méthode ou la traduire en une suite de lectures et d'écritures ce qui nécessiterait de garder sous forme de copies locales toutes les variables (autres que les inputs) dont elles pourraient dépendre. Enfin, il serait possible de s'inspirer des techniques de synthèse de moniteurs proposées par Falcone *et al.* [Falcone et al., 2008, Falcone et al., 2009, Falcone et al., 2012], Koucham *et al.* [Koucham et al., 2016] ou encore Guiochet et Powell [Guiochet and Powell, 2005] pour la génération des fichiers de configuration du filtre.

Chapitre 4

Vérification formelle de protocoles de communication industriels

> Without requirements or design, programming is the art of adding bugs to an empty text file.
>
> *(Louis Srygley)*

Résumé du chapitre

Suite à la publication du protocole Needham-Schroeder en 1978 [Needham and Schroeder, 1978], plusieurs travaux ont été menés pour vérifier formellement la sécurité de protocoles cryptographiques tels que SSH et TLS. Il apparaît que très peu de protocoles de communication industriels sont prévus pour apporter de la sécurité et ceux qui le sont ne sont pas vérifiés formellement. Dans ce chapitre, nous proposons deux contributions dans le but d'adapter les techniques de vérification de protocoles cryptographiques aux protocoles de communication industriels et à leurs contextes. Dans un premier temps, nous réalisons une analyse de propriétés de secret et d'authentification sur les protocoles de handshake d'OPC-UA à l'aide de l'outil ProVerif. Ensuite, nous introduisons une formalisation de propriétés basées sur l'ordre des messages, actuellement inexistantes dans les outils de vérification. Nous appelons cette classe de propriété *intégrité du flux* et la vérifions sur les protocoles MODBUS et OPC-UA à l'aide de l'outil Tamarin.

Sommaire

4.1. Contexte

Comme explicité en section 2.3.2, les automates programmables utilisés durant l'attaque Maroochy Shire communiquaient via les protocoles de communication industriels MODBUS et DNP3. Ces deux protocoles, comme beaucoup d'autres du domaine, ne fournissent aucune protection contre des attaquants. Un attaquant capable de communiquer avec les automates peut donc lancer arbitrairement des commandes affectant le procédé. En particulier, il peut se faire passer pour le client et lancer une commande afin, par exemple, d'arrêter la pompe lorsque la cuve est pleine. Pour pallier ce risque, plusieurs travaux académiques ont proposé de renforcer ces protocoles en ajoutant de la sécurité à certains protocoles tels que MODBUS [Fovino et al., 2009, Hayes and El-Khatib, 2013]. Le protocole OPC-UA a notamment été développé à partir de son prédécesseur OPC et implémente des mesures de sécurité qui se veulent à l'état de l'art des attaques. Cependant, comment s'assurer que les solutions proposées peuvent réellement contribuer à empêcher les attaques ? La vérification de protocoles cryptographiques consiste à appliquer des méthodes formelles sur la spécification de tels protocoles afin de tester des propriétés de sécurité. Ce chapitre s'intéresse à porter et étendre les avancées de ce domaine aux protocoles de communication industriels en vérifiant le protocole de handshake d'OPC-UA et en spécifiant une nouvelle classe de propriétés adaptées aux systèmes industriels.

4.1.1. Besoin de vérifications formelles

La vérification formelle consiste à prouver ou réfuter la validité d'une propriété en utilisant des méthodes formelles issues des mathématiques. Il est difficile de dater les premiers travaux tellement de thématiques peuvent en faire partie. Ces vérifications ont cependant énormément gagné en popularité parmi le public académique dans les années 80 et commencé à se répandre dans l'industrie dans les années 90 notamment à la suite du bug FDIV de Pentium [Sharangpani and Barton, 1994, Nicely, 1995] puis du crash de la fusée Ariane 5 le 4 juin 1996 [Lions, 1996]. Les vérifications formelles s'appliquent autant aux spécifications qu'aux codes sources ou encore aux binaires compilés.

Il va sans dire que les vérifications formelles devraient donc être appliquées à chaque étape du processus de développement. Cependant, c'est rarement le cas pour des questions de budget. Aussi, alors qu'ils font l'effort de vérifier formellement un produit, les industriels sont souvent obligés de choisir entre vérifier les spécifications et le produit fini. Les deux possibilités présentent des avantages et des inconvénients. En effet, vérifier formellement le produit fini a l'avantage de garantir que la vérification ne sera pas invalidée par le reste du développement ou la compilation. Par contre, en cas de problème trouvé, c'est souvent toute la chaîne de production qu'il faut revoir, ce qui implique une perte de temps et des coûts accrus. À l'opposé, vérifier formellement les spécifications à l'intérêt majeur de découvrir les problèmes sur le produit et de les corriger avant que celui-ci ne soit conçu. Dans tous les cas, il est essentiel de faire ces vérifications avant la mise sur le marché du produit afin d'éviter tout risque de rappel, très grave financièrement et pour l'image de marque. Ce besoin a notamment été mis en avant en 2006 par Igure *et al.* [Igure et al., 2006] et en 2009 par Patel *et al.* [Patel et al., 2009]. Ils soutiennent en particulier que la vérification des protocoles aide à découvrir et comprendre la plupart des vulnérabilités d'un protocole avant la publication de son standard et permet de limiter le nombre de révisions qui coûtent du temps et de l'argent.

4.1.2. Vérification de protocoles cryptographiques

Un protocole cryptographique est une succession d'échanges de messages impliquant des primitives cryptographiques. Ces protocoles peuvent avoir des buts variés tels que la distribution de clés, le paiement en ligne ou encore le vote. On peut par exemple penser à SSH et TLS. La vérification de protocoles cryptographiques consiste à décrire ces protocoles de façon formelle et à vérifier si les propriétés qu'ils apportent sont bien garanties (par exemple « un électeur ne peut pas voter deux fois »). Ce domaine a commencé par des preuves manuelles. En particulier à la suite du protocole d'authentification Needham-Schroeder en 1978 [Needham and Schroeder, 1978], Burrows, Abadi et Needham proposent en 1989 la logique BAN [Burrows et al., 1989] (du nom de ses créateurs). Ils utilisent notamment cette logique pour prouver des propriétés de sécurité sur le protocole Needham-Schroeder. Attardons-nous un instant sur une version simplifiée[1] de ce protocole décrite en figure 4.1. *Alice* envoie un premier message à *Bob* contenant son identité (A) et un nombre aléatoire fraîchement généré appelé un *nonce* (N_a), le tout chiffré avec la clé publique de *Bob* (pk_b). *Bob* répond en renvoyant le nonce envoyé par *Alice* (N_a), chiffré cette fois avec la clé publique de *Alice* (pk_a). Ce faisant

1. http://www.fil.univ-lille1.fr/~hym/e/svl-07/needham-schroeder.html

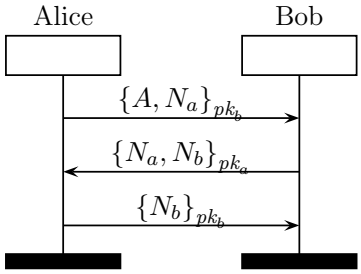


FIGURE 4.1. – Protocole Needham-Schroeder (version simplifiée)

il garantit son identité puisqu'il est le seul à pouvoir déchiffrer le message précédent. Il joint aussi son propre nonce (N_b) qu'il a généré. Enfin, *Alice* renvoie le nonce de *Bob* afin de prouver également son identité. À la fin du protocole, *Alice* et *Bob* sont certains de se parler l'un à l'autre (on parle d'authentification mutuelle). La version complète de ce protocole suppose que *Alice* et *Bob* ne connaissent initialement pas la clé publique de l'autre et doivent demander ces clés à un serveur tiers. La version simplifiée du protocole a été prouvée en 1990 par les auteurs de la logique BAN dans l'article où ils présentent cette logique. Cependant en 1995, soit 17 ans après la publication du protocole en 1978, Gavin Lowe publie une attaque contre le protocole Needham-Schroeder, trouvée en utilisant le model-checker FDR [Roscoe, 1994, Gibson-Robinson et al., 2014]. Cette attaque, montrée en figure 4.2 est par la suite qualifiée de Man-In-The-Middle [Lowe, 1995].

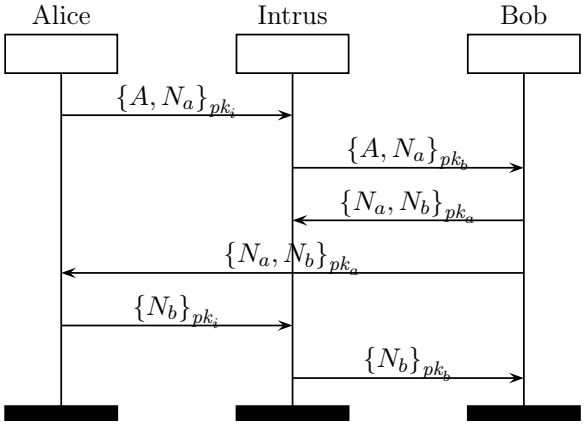


FIGURE 4.2. – Attaque contre le protocole Needham-Schroeder [Lowe, 1995]

L'attaque trouvée par Lowe consiste à jouer le protocole dans deux sessions (deux exécutions), l'une entre *Alice* et l'intrus et l'autre entre l'intrus et *Bob*. En soit, les deux sessions respectent le protocole à la lettre et c'est l'utilisation de messages d'une session dans l'autre et leur entrelacement qui permet à l'intrus de se faire passer pour *Alice* auprès de *Bob*. En 1996, Lowe propose notamment une correction qui consiste à ajouter l'identité de *Bob* dans le second message de façon à ce qu'*Alice* s'aperçoive que le message en question vient de *Bob* et non de l'intrus [Lowe, 1996]. Lowe vérifie au passage la validité de sa correction avec FDR.

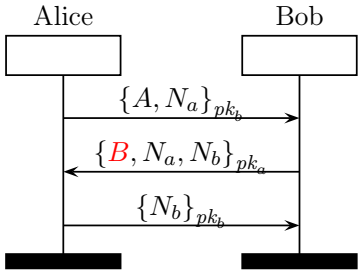


FIGURE 4.3. – Protocole Needham-Schroeder corrigé par Lowe [Lowe, 1995]

La preuve manuscrite avec la logique BAN n'était pourtant pas fausse. Elle ne prenait simplement pas en compte le fait qu'un intrus puisse jouer plusieurs sessions du protocole en parallèle et réutiliser les messages d'une session dans d'autres. Cependant, étudier tous les entrelacements possibles entre plusieurs sessions dépasse de loin les capacités du cerveau humain. En particulier, sans même prendre en compte le fait qu'un message d'une session puisse être rejoué dans d'autres, le nombre possibles d'entrelacements pour n sessions d'un protocole de m messages est $\frac{(m \cdot n)!}{(m!)^n}$ possibilités [2]. Ainsi à titre d'exemple, pour tester les trois messages du protocole Needham-Schroeder dans toutes les sessions possibles entre *Alice*, *Bob* et l'intrus (6 sessions au total), il y aurait déjà 137.225.088.000 entrelacements possibles, encore une fois, sans même prendre en compte le fait qu'un message d'une session puisse être rejoué dans d'autres, ni que l'intrus puisse en forger d'autres. À titre d'exemple, les protocoles de handshake de SSH et TLS (utilisé dans HTTPS) comptent respectivement six et huit messages échangés et sont de fait confrontés à l'explosion combinatoire.

4.1.3. Une multitude d'outils

Pour pallier l'explosion combinatoire induite par les entrelacements de sessions, de nombreux outils ont été proposés depuis la fin des années 90. Ces outils ont tous la particularité que l'attaquant est « codé en dur » dans l'outil et sont ainsi optimisés pour ce modèle d'attaquant. Cet attaquant se nomme intrus Dolev-Yao [Dolev and Yao, 1981] et est décrit en section 4.2.1. Il peut notamment intercepter les messages envoyés sur le réseau, les bloquer, les modifier, les rejouer ou encore en forger de nouveau en fonction de ses connaissances. La cryptographie est par contre supposée sûre, ce qui signifie que l'intrus ne peut pas déchiffrer un message ou forger une signature sans la clé associée. Or, comme expliqué en section 4.2.3, la vérification de protocoles cryptographiques en présence de l'intrus Dolev-Yao n'est pas décidable dans le cas général. Ainsi, certains outils font de la preuve en sur-approximant les capacités de l'intrus. En l'absence d'attaque trouvée par l'outil, on est certain de la sécurité du protocole mais en cas d'attaques trouvées, on ne sait pas si elles viennent de l'approximation. A l'opposé, certains outils font du test en sous-approximant les capacités de l'attaquant. On est alors certains que les attaques trouvées sont réelles mais on peut aussi en manquer. Parmi les outils de vérification de protocoles cryptographiques les plus connus, on peut notamment citer :

2. https://math.stackexchange.com/questions/77721/number-of-instruction-interleaving

NRL Développé en 1996 par Catherine Meadows [Meadows, 1996], NRL (pour *United States Naval Research Laboratory*) est considéré comme le premier outil de vérification de protocoles cryptographiques. L'outil, comme les protocoles à analyser, sont écrits en Prolog [Roussel, 1975]. Il sous-approxime le comportement de l'attaquant en fixant le nombre de sessions considérées.

Casper/FDR[3] Développé en 1998 par Gavin Lowe [Lowe, 1998] à la suite des ses travaux sur le protocole Needham-Schroeder, il permet de générer automatiquement un modèle pouvant être analysé par le model-checker FDR. Pour cela, l'outil sous-approxime le comportement de l'attaquant pour un nombre fixé de sessions.

SPEAR II Développé en 2013 par Saul et Hutchison [Saul and Hutchison, 1999], SPEAR II est un outil qui permet la spécification et la vérification de protocoles cryptographiques. Cependant il ne se concentre pas que sur la sécurité des protocoles, en proposant par exemple des analyses de performances et de la génération de code. Les vérifications de sécurité sont menées via la logique BAN ayant servi à prouver le protocole Needham-Schroeder en 1990.

CL-Atse[4] Développé en 2006 par Mathieu Turuani [Turuani, 2006], CL-Atse (pour *Constraint-Logic-based Attack Searcher*) étudie les protocoles en énumérant les traces possibles et en les transformant en contraintes qui peuvent ensuite être résolues. L'outil sous-approxime le comportement de l'attaquant pour un nombre fixé de sessions. Il supporte les théories équationnelles Diffie-Hellman et XOR.

OFMC[5] Développé en 2003 et maintenu par Sebastian Mördersheim [Basin et al., 2003], OFMC (pour *Open-source Fixed-point Model-Checker*) effectue du model-checking symbolique pour un nombre borné de sessions. Il sous-approxime le comportement de l'attaquant pour un nombre fixé de sessions et supporte les théories équationnelles Diffie-Hellman et XOR.

SAT-MC[6] Développé en 2004 par Alessandro Armando et Luga Compagna [Armando and Compagna, 2005, Armando et al., 2014], il se base à la fois sur le model-checker NuSMV et le SAT-solver MiniSAT pour vérifier un protocole. Pour cela, l'outil sous-approxime le comportement de l'attaquant pour un nombre fixé de sessions. SAT-MC est notamment utilisé par l'outil Tookan [Bortolozzo et al., 2010] utilisé par Steel *et al.* pour vérifier les HSM (*Hardware Security Module*) PKCS #11.

TA4SP[7] Développé en 2004 par Yohan Boichut [Boichut et al., 2004], TA4SP (pour *Tree Automata based on Automatic Approximations for the Analysis of Security Protocols*) est basé sur l'outil de réécriture Timbuk développé par Thomas Genet [Genet, 1998]. Il

3. http://www.cs.ox.ac.uk/gavin.lowe/Security/Casper/
4. http://webloria.loria.fr/equipes/cassis/softwares/AtSe/
5. http://www.imm.dtu.dk/~samo/
6. https://www.ai-lab.it/software/satmc/index.html
7. http://www.univ-orleans.fr/lifo/membres/Yohan.Boichut/ta4sp.html

est capable de faire aussi bien de la sur-approximation que de la sous-approximation et peut analyser le protocole pour un nombre non borné de sessions. Cependant, il ne gère que des propriétés de secrets, là où les autres outils vérifient aussi l'authentification. Par ailleurs, aucune trace n'est reconstruite lorsqu'une attaque est trouvée.

CL-Atse, OFMC, SAT-MC et TA4SP sont des outils internes utilisés par la suite AVISPA [Armando et al., 2005] (pour *Automated Validation of Internet Security Protocols and Applications*). Tous ces outils ont donc un langage d'entrée commun appelé HLPSL pour (*High Level Protocol Specification Language*).

ProVerif[8] Développé en 2001, ProVerif [Blanchet, 2001, Blanchet et al., 2017] est aujourd'hui toujours maintenu par Bruno Blanchet. C'est un outil de vérification qui analyse un nombre non borné de sessions en utilisant des techniques de sur-approximation. D'abord uniquement modélisés en clauses de Horn [Horn, 1951], les protocoles peuvent maintenant être écrits dans un sous-ensemble du π-calcul [Milner et al., 1992] (alors retransformés en clauses de Horn par l'outil). Il permet notamment la définition de fonctions, types et théories équationnelles utilisables dans la modélisation et en particulier diverses primitives cryptographiques (notamment Diffie-Hellman). Les propriétés sont exprimées sous forme d'implication d'événements pouvant être levés dans le modèle. Lorsqu'une attaque est trouvée, l'outil tente de reconstruire une trace menant à l'attaque.

Scyther[9] Développé en 2008 par Cas Cremers [Cremers, 2008], il vérifie des protocoles pour un nombre borné ou non borné de sessions et a la particularité de garantir la terminaison en bornant l'analyse. Scyther utilise des techniques de sur-approximation et ne supporte pas de base des propriétés algébriques comme XOR et Diffie-Hellman mais il est possible d'y sous-approximer Diffie-Hellman en ajoutant des processus implémentant la théorie équationnelle [Cremers, 2011]. Une autre particularité de Scyther est que d'automatiser l'expression des propriétés d'authentification à vérifier.

Tamarin[10] Développé en 2012 durant les thèses de Simon Meier et Benedikt Schmidt [Schmidt et al., 2012, Meier et al., 2013], il est maintenant maintenu en collaboration entre l'ETH Zurich, l'Université d'Oxford et le Loria Nancy. Comme ProVerif, il est capable de vérifier des protocoles pour un nombre non borné de sessions. Les protocoles sont spécifiés sous forme de règles de réécriture portant sur des *multisets* et les propriétés sont spécifiées sous forme de formules de logique temporelle du premier ordre. Tamarin se base sur l'outil de réécriture Maude [Clavel et al., 1996] et supporte Diffie-Hellman nativement.

Dans un article publié en 2015 à la conférence FPS 2015 [Lafourcade and Puys, 2015], nous avons réalisé un benchmark des principaux outils capables de raisonner sur des protocoles avec des propriétés algébriques comme l'exponentiation Diffie-Hellman ou le XOR. Les outils listés ci-dessus s'intéressent à des propriétés d'accessibilité comme le

8. http://prosecco.gforge.inria.fr/personal/bblanche/proverif/
9. https://www.cs.ox.ac.uk/people/cas.cremers/scyther/
10. http://www.infsec.ethz.ch/research/software/tamarin.html

secret ou l'authentification. On cherche donc à atteindre un état dans lequel la propriété est violée. Cependant certains outils comme ProVerif, Tamarin et récemment AKISS permettent de vérifier des propriétés d'équivalence observationnelle. Ces propriétés cherchent à voir si l'intrus est en mesure de distinguer deux traces. C'est particulièrement intéressant pour les protocoles de vote en ligne où l'on cherche à vérifier si l'intrus peut déterminer le vote de quelqu'un.

4.1.4. État de l'art des vérifications de protocoles industriels

Les premiers travaux vérifiant la sécurité des protocoles de communication industriels se sont limités à discuter des fonctionnalités apportées ou non par les protocoles. En 2004, Clarke *et al.* [Clarke et al., 2004] ont discuté de la sécurité des protocoles DNP3 (*Distributed Network Protocol*) et ICCP (*Inter-Control Center Communications Protocol*). En 2005, Dzung *et al.* [Dzung et al., 2005] ont étudié informellement les fonctionnalités apportées par plusieurs protocoles industriels, notamment : OPC (*Open Platform Communications*), MMS (*Manufacturing Message Specification*), IEC/CEI 61850, ICCP et Ethernet/IP. En 2006, les auteurs de la documentation technique d'OPC-UA (*OPC Unified Architecture*) détaillent les fonctions de sécurité devant être implémentées dans les piles réseaux du protocole. En 2009, Fovino *et al.* [Fovino et al., 2009] proposent une version sécurisée de MODBUS utilisant des primitives cryptographiques et des fonctions de hachage telles que RSA et SHA2. En 2013, Hayes et El-Khatib [Hayes and El-Khatib, 2013] proposent à leur tour une version sécurisée du protocole MODBUS, celle-ci basée sur des codes d'authentification de message (MAC– *Message Authentication Code*) et se basant sur le protocole de communication STCP au lieu de TCP. Enfin en 2015, Wanying *et al.* [Wanying et al., 2015] résument les fonctionnalité de sécurité offertes par les protocoles MODBUS, DNP3 et OPC-UA, concluant qu'OPC-UA est plus sûr car il utilise des primitives cryptographiques. Cependant, aucun des travaux ci-dessus ne prouvent formellement leurs conclusions sur la sécurité (ou l'absence de sécurité) des protocoles étudiés.

Dans les travaux plus récents, des analyses formelles commencent à apparaître dans les études de sécurité de protocoles de communication industriels. En 2007, Patel et Yu [Patel and Yu, 2007], proposent une vérification formelle du protocole DNP3 à l'aide des outils OFMC et SPEAR II. En 2008, Dutertre [Dutertre, 2007] introduit des spécifications formelles pour vérifier si une implémentation respecte la syntaxe et la sémantique de MODBUS en utilisant PVS, un prouveur de théorèmes. Cette contribution n'a cependant rien à voir avec la sécurité du protocole mais vise à aider les développeurs à vérifier la conformité de leurs implémentations face aux standards. Enfin en 2016, Amoah [Amoah, 2016] utilise les réseaux de Petri afin de vérifier formellement des propriétés d'authentification sur le protocole DNP3.

Contributions : Dans le cadre de cette thèse, nous proposons deux contributions à la vérification formelle de protocoles de communication industriels :

— Premièrement une analyse de propriétés de secret et d'authentification sur les handshake d'OPC-UA à l'aide de l'outil ProVerif en modélisant le protocole à partir du standard. Ces travaux ont donné lieu à une publication à la conférence SAFECOMP 2016 [Puys et al., 2016a] ;

— Deuxièmement une formalisation de propriétés basées sur l'ordre des messages, actuellement inexistantes dans les outils de vérification mais cruciale pour les systèmes industriels. Ces travaux ont donné lieu à une publication à la conférence SECRYPT 2017 [Dreier et al., 2017b]. Nous appelons cette classe de propriété *intégrité du flux* et l'implémentons dans l'outil Tamarin.

La table 4.1 résume les différents travaux présentés ci-dessus en incluant les contributions de cette thèse.

Référence	Année	Protocoles étudiés	Type d'analyse
[Clarke et al., 2004]	2004	DNP3, ICCP	Informelle
[Dzung et al., 2005]	2005	OPC, MMS, IEC/CEI 61850 ICCP, Ethernet/IP	Informelle
[Graham and Patel, 2004]	2004	DNP3	Formelle (OFMC)
[IEC-62541, 2015]	2006	OPC-UA	Informelle
[Patel and Yu, 2007]	2007	DNP3	Informelle
[Fovino et al., 2009]	2009	MODBUS	Informelle
[Hayes and El-Khatib, 2013]	2013	MODBUS	Informelle
[Wanying et al., 2015]	2015	MODBUS, DNP3, OPC-UA	Informelle
[Amoah, 2016]	2016	DNP3	Formelle (Réseaux de Petri)
[Puys et al., 2016a]	2016	OPC-UA	Formelle (ProVerif)
[Dreier et al., 2017b]	2017	MODBUS, OPC-UA	Formelle (Tamarin)

TABLE 4.1. – État de l'art des vérifications de protocoles industriels

Plan du chapitre : Dans un premier temps, la section 4.2 discute des difficultés liées à la vérification de protocoles cryptographiques. Ensuite, la section 4.3 présente nos travaux sur la vérification formelle du handhake d'OPC-UA. La section 4.4 décrit notre formalisation des propriétés d'intégrité du flux. Enfin, la section 4.5 conclut et dresse les perspectives des contributions.

4.2. Difficultés liées à la vérification de protocoles cryptographiques

Comme avancé en section 4.1, plusieurs difficultés interviennent dans la vérification de protocoles cryptographiques et notamment le fait que plusieurs paramètres sont non bornés et en particulier le comportement de l'attaquant. On commence par présenter l'intrus Dolev-Yao en section 4.2.1 ainsi que les propriétés classiques vérifiées par les outils en section 4.2.2. Ensuite, on discute des sources potentielles d'indécidabilité en section 4.2.3 et on s'intéresse en section 4.2.4 aux approximations faites par les outils pour rendre le problème décidable. Enfin, on présente la possibilité d'étendre le modèle Dolev-Yao et les conséquences que cela engendre en section 4.2.5.

4.2.1. L'intrus Dolev-Yao

L'intrus Dolev-Yao a été introduit par Dolev et Yao en 1981 [Dolev and Yao, 1981]. Il formalise un attaquant contrôlant le réseau et accédant à tous les messages envoyés,

le réseau étant en général représenté par un canal de communication sur lequel tous les participants lisent et écrivent. Tous les messages passant par ce canal peuvent être mémorisés par l'intrus. Tous les messages reçus par les participants sur ce canal sont également modélisés comme envoyés par l'intrus, il peut donc les bloquer, les modifier, les rejouer ou encore en forger de nouveau en fonction de ses connaissances. Ces dernières sont constituées de ses connaissances initiales T (par exemple l'identité des participants ou leurs clés publiques) et des messages mémorisés. L'intrus Dolev-Yao possède également un *système de déduction* lui permettant de déduire de nouvelles connaissances à partir de celles emmagasinées jusque là. Dans ce système de déduction, les messages échangés sont composés de termes construits sur la signature logique :

$$\Sigma = \{\langle u, v\rangle, \{u\}_v\} \cup \mathcal{S}_{fun}$$

avec $\langle u, v\rangle$ désignant le couple des termes u et v, $\{u\}_v$ le chiffré de u avec la clé v [11], et $\mathcal{S}_{fun}$ un ensemble de fonctions accessibles à l'intrus. On note $T \vdash u$ lorsque l'intrus est capable de déduire le terme u à partir de ses connaissances T. Le système de déduction de l'intrus Dolev-Yao est constitué des règles de déduction présentées en figure 4.4 avec :

- (A) : l'intrus peut utiliser tout terme de ses connaissances T ;
- (P) : l'intrus peut construire un couple de messages ;
- (PG) et (PD) : l'intrus peut extraire la partie gauche (resp. droite) d'un couple ;
- (C) et (D) : l'intrus peut chiffrer (resp. déchiffrer) un terme à l'aide d'une clé ;
- (F) : l'intrus peut construire un terme en appliquant une fonction $f \in \mathcal{S}_{fun}$ (par exemple une fonction de hachage).

$$(A)\quad \frac{u \in T}{T \vdash u} \qquad (C)\quad \frac{T \vdash u \quad T \vdash k}{T \vdash \{u\}_k}$$

$$(P)\quad \frac{T \vdash u \quad T \vdash v}{T \vdash \langle u, v\rangle} \qquad (D)\quad \frac{T \vdash \{u\}_k \quad T \vdash k}{T \vdash u}$$

$$(PG)\quad \frac{T \vdash \langle u, v\rangle}{T \vdash u} \qquad (F)\quad \frac{T \vdash u \quad T \vdash f}{T \vdash f(u)}$$

$$(PD)\quad \frac{T \vdash \langle u, v\rangle}{T \vdash v}$$

FIGURE 4.4. – Système de déduction de l'intrus Dolev-Yao

Ce modèle d'intrus est souvent considéré comme très puissant de part son contrôle total du réseau, la multitude d'actions qu'il peut exécuter et son système de déduction. Les capacités de l'intrus ont cependant des limites. En particulier, la cryptographie est supposée sûre. Cela signifie que l'intrus ne peut pas déchiffrer un message ou forger

11. La règle est ici définie pour des primitives symétriques mais peut aisément être étendue à des primitives asymétriques.

une signature sans la clé associée. Cette hypothèse est nommée *Perfect Encryption Assumption*.

4.2.2. Propriétés vérifiables

Comme décrit en section 4.1, les propriétés historiquement vérifiées en vérification de protocoles cryptographiques sont des propriétés d'accessibilité. On veut vérifier s'il est possible que le système, comprenant l'ensemble des participants communiquant en appliquant le protocole, puisse atteindre un état spécifié. Deux propriétés sont notamment considérées : le secret et l'authentification.

Propriété de secret : Le secret (aussi appelé la confidentialité) vise à vérifier s'il existe un chemin menant à un état où l'intrus aurait connaissance d'un certain terme (par exemple une clé cryptographique). Cette propriété peut être globale ou locale. Dans le second cas, on souhaite vérifier que l'intrus n'apprend pas un terme secret pendant l'exécution d'une session du protocole.

Propriété d'authentification : L'authentification a été formalisée par plusieurs travaux [Lowe, 1997, Schneider, 1998] et cette propriété peut exprimer des notions plus ou moins fortes. La notion la plus faible introduite par Lowe se nomme *aliveness*. Elle consiste à vérifier si, lorsque *Alice* joue un protocole apparemment avec *Bob*, alors *Bob* a effectivement joué le protocole. Une notion plus forte est nommée *weak agreement* et ajoute la condition que *Bob* ait apparemment joué le protocole avec *Alice*. Des notions plus forte impliquent que les participants s'authentifient sur des messages. *Alice* pourra par exemple prouver son identité en signant un message avec sa clé privée qu'elle est supposément la seule à posséder. Lorsque *Bob* reçoit le message signé par *Alice*, on souhaite vérifier si c'est bien *Alice* qui l'a envoyé à *Bob* ou si l'intrus a pu le rejouer ou le signer à la place d'*Alice*. Concrètement, la trace d'exécution (dans laquelle plusieurs sessions s'entrelacent) est annotée par des événements. En particulier, un événement est levé lorsque *Alice* envoie le message m à *Bob* (par exemple noté $\mathit{Envoi}(\mathit{Alice}, \mathit{Bob}, m)$). Un autre événement est levé lorsque *Bob* reçoit le message m' venant de *Alice* (par exemple noté $\mathit{Reception}(\mathit{Bob}, \mathit{Alice}, m')$). On cherche alors à vérifier s'il existe une correspondance entre ces événements (pour tout événement $\mathit{Reception}(Y, X, m')$, il doit exister un événement $\mathit{Envoi}(X, Y, m)$ avec $m = m'$ le précédent). Cette propriété est nommée authentification non injective (*non-injective agreement*). Une version plus forte consiste à vérifier si le nombre d'événements de réception correspond au nombre d'événements d'envoi. On parle alors d'authentification injective (*injective agreement*).

Certain travaux ont montré comment exprimer des propriétés plus complexes au moyen de chiffrement et d'authentification. C'est par exemple le cas de la non-répudiation [Kremer and Raskin, 2001]. Une autre classe de propriété se nomme l'équivalence observationnelle. Ces propriétés cherchent à voir si l'intrus est en mesure de distinguer deux traces. Les propriétés d'équivalence observationnelle sont souvent considérées comme plus difficile à démontrer et commencent seulement à être intégrées dans certains outils comme ProVerif, Tamarin et plus récemment AKISS.

4.2.3. Sources d'indécidabilité

Comme décrit par Cortier en 2005 [Cortier, 2005], la vérification de protocoles cryptographiques est indécidable dans le cas général. Cela vient du fait que plusieurs paramètres sont non bornés, en particulier :

— le nombre de sessions ;
— le nombre de participants ;
— les messages sont de taille arbitraire (l'intrus peut construire des couples de messages à l'infini) ;
— n'importe quel message de la connaissance de l'intrus peut être envoyé ;
— l'intrus peut générer un nombre arbitraire de nouveaux termes (par exemple des clefs ou des nonces).

De nombreux travaux se sont intéressés à la vérification de protocoles cryptographiques en bornant certains de ces paramètres et obtiennent des résultats de décidabilité, voir de complexité. Par exemple, le problème du secret devient décidable en bornant le nombre de sessions. C'est notamment ce qui est fait dans les outils de la suite AVISPA.

4.2.4. Approximations

Afin de rendre le problème décidable, il est donc nécessaire d'introduire des approximations afin de borner certains paramètres du modèle. Le nombre de sessions peut être borné arbitrairement par l'utilisateur (en « choisissant qui parle avec qui »). Une autre approche consiste à borner le nombre de nonces pouvant être utilisés dans le protocole. Cela peut par exemple se faire en réutilisant des nonces afin de n'en considérer qu'un ensemble fini [Broadfoot et al., 2000] ou alors en remplaçant les nonces par une fonction des messages reçus (comme dans le cas de ProVerif [Blanchet et al., 2017]). Enfin, une cause de non-terminaison peut être que des messages liés à une étape du protocole qui peuvent être utilisés pour une autre (le protocole « boucle »). Ce problème peut être évité en ajoutant une étiquette (ou *tag*) à chaque message afin qu'il ne puisse pas être utilisé à une autre étape [Blanchet and Podelski, 2003]. Parfois, la modélisation elle-même introduit des approximations non désirées. Par exemple, ProVerif traduit les protocoles à analyser en clauses de Horn. Plusieurs limites de cette logique se répercutent alors dans l'analyse du protocole. Par exemple, un message envoyé une fois par un participant peut être reçu un nombre infini de fois sans que l'intrus n'ait besoin de le rejouer. Ces limitations de ProVerif seront discutées plus en détails en section 5.5.1.

4.2.5. Aller au delà des vérifications habituelles

Là où la plupart des outils se limitent à vérifier les propriétés de secret et d'authentification présentées en section 4.2.2, ProVerif et Tamarin permettent à l'utilisateur de spécifier ses propres propriétés sous forme de formules logiques. Cette fonctionnalité sera étudiée plus en détail au chapitre 5 afin d'exprimer des propriétés de sûreté. Ces mêmes outils permettent également de spécifier des théories équationnelles personnalisées en plus de celles de base du modèle Dolev-Yao. Ajouter des théories équationnelles au modèle consiste à augmenter les capacités de l'intrus au même titre que celles des

autres participants. Les théories équationnelles permettent de modéliser des règles de déduction particulières comme les signatures cryptographiques ou encore la vérification de mots de passe, comme nous le montrerons en section 4.3. Par exemple, les théories équationnelles en listing 4.1 décrivent respectivement des signatures symétriques et asymétriques dans un formalisme similaire à celui de ProVerif.

```
(* Signatures symétriques *)
fun symsign(msg, sharedkey): signature
fun symverify(
    msg,
    symsign(msg, sharedkey),
    sharedkey
) = true

(* Signatures asymétriques *)
fun pk(secretkey)
fun asymsign(msg, publickey): signature
fun asymverify(
    msg,
    asymsign(msg, secretkey),
    pk(secretkey)
) = true
```

Listing 4.1 – Théories équationnelles pour les signatures cryptographiques

Une autre utilisation classique des théories équationnelles est la spécification de propriétés algébriques d'opérateurs présents dans le protocole comme le XOR ou l'exponentiation Diffie-Hellman (par exemple la commutativité). Nous avons notamment utilisé la possibilité d'exprimer des théories équationnelles en ProVerif afin de vérifier des protocoles de calculs partagés utilisant des primitives cryptographiques homomorphes. Ces travaux ont donné lieu à des publications à la conférence SECRYPT 2016 et dans le journal *Computers & Security* [Dumas et al., 2016, Dumas et al., 2017]. Le contrecoup de ces théories est une augmentation de la complexité du modèle à analyser qui peut se traduire par de nouvelles approximations (par exemple en ne considérant que certaines propriétés des opérateurs). Les théories équationnelles nous seront particulièrement utile en section 4.4 afin de spécifier des propriétés adaptées aux systèmes industriels et non prévues de base par les outils.

4.3. Vérification formelle du handshake d'OPC-UA

Une première utilisation des outils de vérification de protocoles cryptographiques dans le cadre des systèmes industriels consiste à vérifier des propriétés de secret et d'authentification, classiques pour les outils. En particulier, dans l'article [Puys et al., 2016a], nous nous sommes intéressés à vérifier de telles propriétés sur le protocole permettant de démarrer des sessions OPC-UA. Il s'agit en l'occurrence de deux sous-protocoles nommés *OpenSecureChannel* et *CreateSession*. Le premier vise à authentifier mutuellement le client et le serveur notamment à l'aide de leurs clés asymétriques et à dériver des clés symétriques fraîches utilisées durant le reste de la session. Le second permet ensuite à l'humain présent sur le client de s'authentifier en envoyant ses identifiants. Nous utilisons l'outil ProVerif pour vérifier des propriétés de secret et d'authentification

sur ces deux protocoles en présence d'un intrus Dolev-Yao. L'utilisation des outils de vérification est plutôt classique en terme de protocoles et de propriétés étudiées. Cependant, la modélisation d'identifiants et de mots de passe en ProVerif n'est pas classique et requiert de bien modéliser certaines hypothèses sur l'utilisation des mots de passe par les clients afin d'être juste. Nous modélisons les protocoles tels qu'ils sont décrits dans les standards [IEC-62541, 2015] [12]. Nous commençons par présenter la sémantique de ProVerif en section 4.3.1. Ensuite, nous analysons le protocole *OpenSecureChannel* en section 4.3.2 puis le protocole *CreateSession* en section 4.3.3 et discutons des résultats.

4.3.1. Fonctionnement de ProVerif

Un modèle en ProVerif est constitué de trois parties : (i) les déclarations des primitives qui seront utilisées (par exemple les primitives de chiffrement ou encore les théories équationnelles), (ii) les processus représentant les agents du protocole (i.e. : comment chaque agent interagit avec le réseau), et (iii) le protocole en lui-même (i.e. : comment les agents interagissent entre-eux). Les déclarations peuvent être des types, des constantes (aussi appelées *names*), et des fonctions. Chaque constante et fonction peut être publique (donc connue de l'intrus) ou privée. De plus dans le cas des fonctions, on distingue les constructeurs des destructeurs. Les constructeurs construisent des termes complexes en appliquant les fonctions à d'autres sous-termes (par exemple chiffrement, hachage). A l'opposé, les destructeurs permettent d'exprimer les propriétés des fonctions (par exemple le fait que déchiffrer un chiffré avec la bonne clé donne le message original, d'où le nom de « destructeur » qui élimine le constructeur). Par exemple le listing 4.2 présente les constructeurs *pk* et *aenc*, ainsi que le destructeur *adec* permettant de modéliser un chiffrement asymétrique.

```
(* Chiffrement asymétrique. *)
fun pk/1.
fun aenc/2.
reduc adec(aenc(x, pk(k)), k) = x.
```

Listing 4.2 – Exemple de fonctions pour les signatures cryptographiques

Les processus sont exprimés dans un formalisme proche du π-calcul. Ce formalisme permet d'exprimer des processus communiquant à travers des canaux et s'exécutant de façon concurrente.

$P, Q ::=$	$chan(x).P$	Réception de x sur le canal $chan$
$\mid$	$\overline{chan}\langle x\rangle.P$	Envoi de x sur le canal $chan$
$\mid$	$P \mid Q$	Exécution concurrente de P et Q
$\mid$	$(\nu c)P$	Création d'un nouveau canal c
$\mid$	$!P$	Réplication de P
$\mid$	0	Processus nul

Les opérateurs d'exécution concurrente et de réplications ne sont pas utilisés lors de la définition des processus mais plus tard lors de la définition du protocole. A ces opérations

12. Les modèles utilisés peuvent être trouvés à l'adresse suivante : https://maxime.puys.name/files/phdThesisAssets.tar.bz2

de base, ProVerif ajoute des fonctionnalités comme les tests booléens Si-Alors-Sinon ou encore des *bindings* via le mot-clé let similaire à ceux des langages fonctionnels et permet ainsi de faire du *pattern matching* sur les termes. Par exemple les processus modélisant *Alice* et *Bob* dans le protocole Needham-Schroeder peuvent être exprimés comme dans le listing 4.3.

```
(* Processus de Alice. *)
let Alice =
    (* Générer et envoyer Na. *)
    new Na;
    out(c, aenc((Na, A), pkB));

    (* Recevoir la réponse au challenge de Bob. *)
    in(c, repBob);
    let (=Na, Nb) = adec(repBob, skA) in

    (* Répondre au challenge Bob. *)
    out(c, aenc(Nb, pkB));

(* Processus de Bob. *)
let Bob =
    (* Recevoir le challenge de Alice. *)
    in(c, challAlice);
    let (Na, A) = adec(challAlice, skB) in

    (* Générer et envoyer Nb et répondre au challenge de Alice. *)
    new Nb;
    out(c, aenc((Na, Nb), pkA));
```

Listing 4.3 – Roles pour le protocole Needham-Schroeder

Les interactions entre les agents sont exprimées dans un processus principal. C'est aussi souvent ici que sont crées les clés cryptographiques comme montré en listing 4.4.

```
(* Processus principal. *)
process
    new skA; let pkA = pk(skA) in out(c, pkA);
    new skB; let pkB = pk(skB) in out(c, pkB);
    (!Alice) | (!Bob)
```

Listing 4.4 – Processus principal pour le protocole Needham-Schroeder

Enfin, les propriétés de secret peuvent être exprimées en précisant les termes que l'attaquant doit obtenir comme en listing 4.5. Elles peuvent aussi être exprimées sous forme d'implications d'événements qui sont levés dans les processus comme expliqué en section 4.2.2.

```
(* Définition des secrets à trouver par l'attaquant. *)
query attacker:Na.
query attacker:Nb.
```

Listing 4.5 – Objectifs pour le protocole Needham-Schroeder

Pour les définitions formelles de la sémantique de ProVerif, se référer à l'article présentant l'outil [Blanchet, 2001] et au manuel d'utilisation [Blanchet et al., 2017].

4.3.2. OPC-UA *OpenSecureChannel*

Le sous-protocole *OpenSecureChannel* vise à authentifier mutuellement le client et le serveur notamment à l'aide de leurs clés asymétriques et à dériver des clés symétriques fraîches utilisées durant le reste de la session. Pour cela, deux nonces seront échangés (comme dans le protocole Needham-Schroeder). Par ailleurs, comme expliqué en section 2.2.3.2, trois *modes de sécurité* de sécurité sont possibles en OPC-UA. Le mode utilisé déterminera le niveau de sécurité qui sera utilisé dans le reste de la session. Le mode utilisé est choisi par le client parmi ceux proposés par le serveur. Plus précisément les modes de sécurité d'OPC-UA sont :

— *None* qui n'apporte aucune sécurité (par compatibilité avec les matériels trop anciens ou ayant une trop faible puissance pour faire de la cryptographie) ;

— *Sign* où les messages sont signés mais envoyés en clair ;

— *SignEncrypt* où les messages sont signés et chiffrés.

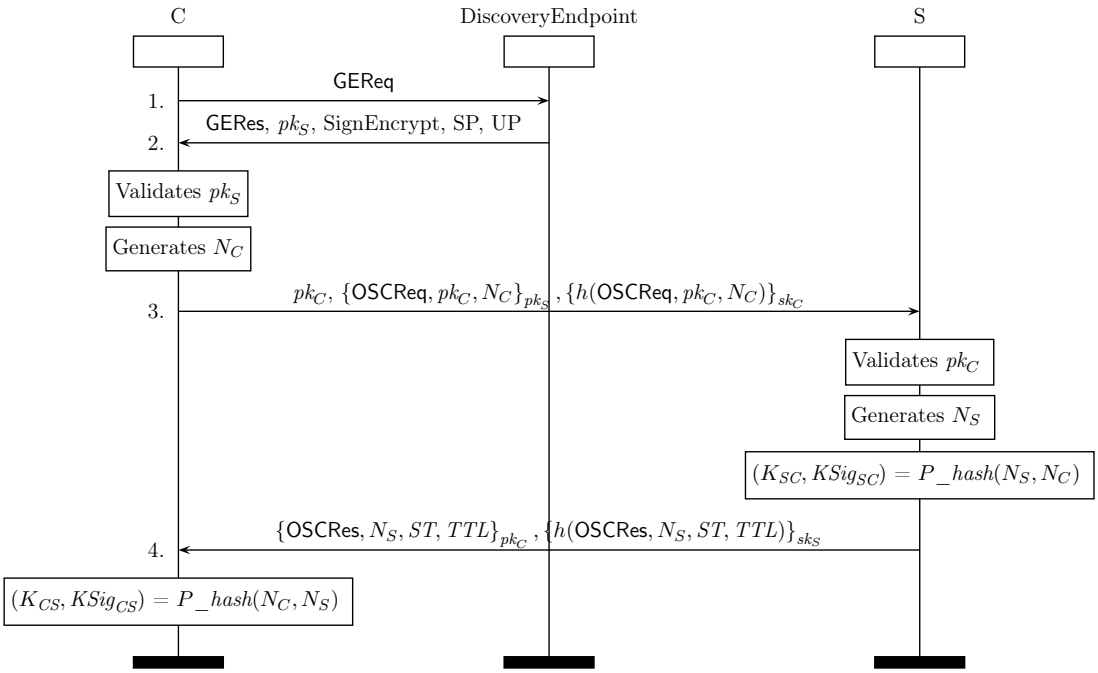


FIGURE 4.5. – Le sous-protocole OPC-UA *OpenSecureChannel* en mode SignEncrypt

Le sous-protocole *OpenSecureChannel* est présenté en figure 4.5. Trois participants sont présents : (i) C le client, (ii) S le serveur, et (iii) *DiscoveryEndoint* un tiers donnant au client la liste des serveurs et leurs modes de configuration. Dans le message 1, C (le client) demande donc des informations sur S (le serveur) avec une requête de type GEReq

pour *GetEndpointRequest*. Dans le message 2, le *DiscoveryEndpoint* répond avec : GERes pour *GetEndpointResponse*, *SP* pour *Security Policy* et *UP* pour *UserPolicy*. *SP* et *UP* sont utilisés pour négocier les primitives cryptographiques utilisées (par exemple RSA ou bien courbes elliptiques). En message 3, C envoie un nonce N_C à S avec OSCReq pour *OpenSecureChannelRequest*. Enfin en message 4, S répond à C avec son propre nonce N_S et OSCRes pour *OpenSecureChannelResponse*, *ST* pour *SecurityToken* (un identifiant unique pour le canal utilisé) et *TTL* pour *TimeToLive* (sa durée de vie). Les quatre termes GEReq, GERes, OSCReq et OSCRes servent à indiquer le but de chaque message. À la fin de ce sous-protocole, C et S utilisent les nonces (N_C et N_S) qu'ils ont générés et reçus afin de dériver quatre clés symétriques : K_{CS}, $KSig_{CS}$, K_{SC} et $KSig_{SC}$. Pour cela ils hachent les nonces avec une fonction nommée P_hash, similaire à celle utilisée dans TLS [Dierks and Rescorla, 2008] : $(K_{CS}, KSig_{CS}) = P\_hash(N_C, N_S)$ et $(K_{SC}, KSig_{SC}) = P\_hash(N_S, N_C)$.

4.3.2.1. Modélisation avec ProVerif

Nous avons choisi l'outil ProVerif car le sous-protocole *CreateSession*, modélisé en section 4.3.3.1, requiert de pouvoir spécifier des théories équationnelles particulières. Pour modéliser le sous-protocole *OpenSecureChannel*, nous faisons quelques hypothèses classiques sur la modélisation des certificats. En particulier, les certificats sont modélisés avec les clé publiques des agents. Nous considérons que l'intrus est capable de fournir un certificat de son identité (donc une clé publique) qui serait accepté par les clients et serveurs. Cet intrus peut par exemple représenter un équipement ayant été corrompu par un virus ou contrôlé par un opérateur malicieux. De plus, lorsqu'un client envoie une *GetEndpointRequest* concernant un serveur, il reçoit généralement une liste de configurations possibles avec différents modes de sécurité. Nous supposons qu'une seule est renvoyée et que le client l'accepte forcément. Pour compenser cette hypothèse, nous vérifions toutes les configurations possibles.

De plus, il n'est pas évident d'après le standard [IEC-62541, 2015][13] si les nonces échangés en mode Sign sont chiffrés ou non (puisque par définition, le mode Sign n'utilise pas de chiffrement). Une pratique courante dans cette situation est d'utiliser un mécanisme de *KeyWrapping* permettant de chiffrer des clés ou des nonces pendant un échange. Cependant, les phrases « A la place, l'algorithme *AsymmetricKeyWrapAlgorithm* est utilisé pour chiffrer les clé symétriques »[14] (Partie 6, 6.1) et « L'élément *AsymmetricKeyWrapAlgorithm* de la structure *SecurityPolicy* [...] n'est pas utilisé pour les implémentations UASC »[15] (Partie 6, 6.7.2) (UASC pour *UA Secure Channel*) semblent en contradiction. Nous choisissons de tester le protocole avec et sans protection des nonces afin de vérifier l'importance de cette fonctionnalité. La même question se pose pour la présence de l'identité des destinataires dans les signatures. Suite à la phrase « Si une empreinte n'est pas spécifiée, cette valeur [identité du destinataire] peut être fixée à 0

13. https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-6-mappings/ Version 1.03 du 20/07/2015.

14. Traduction de « Instead, they use the AsymmetricKeyWrapAlgorithm to encrypt a symmetric key ».

15. Traduction de « The AsymmetricKeyWrapAlgorithm element of the SecurityPolicy [...] is not used by UASC implementations. »

ou -1 » [16] dans la table 27 de la partie 6 de la norme, nous avons choisi d'analyser le protocole avec et sans l'identité du receveur dans les signatures.

4.3.2.2. Résultats

Comme décrit ci-dessus, nous avons modélisé le sous-protocole *OpenSecureChannel* avec ProVerif pour les trois modes de sécurité possibles d'OPC-UA. Nous avons vérifié les objectifs d'analyse suivants : (i) le secret des clés dérivées par C (K_{CS} et $KSig_{CS}$), (ii) le secret des clés dérivées par S (K_{SC} et $KSig_{SC}$), (iii) l'authentification de C par S avec N_C et (iv) l'authentification de S par C avec N_S. Les résultats sont proposés en table 4.2 où ✓ signifie que la propriété est vérifiée et ✗ qu'une attaque a été trouvée.

Mode de sécurité OPC-UA	**Objectifs**			
	Sec K_{CS}	Sec K_{SC}	Auth N_S	Auth N_C
None	✗	✗	✗	✗
Sign	✗	✗	✗	✗
SignEncrypt	✓	✓	✗	✗

TABLE 4.2. – Résultats pour le sous-protocole *OpenSecureChannel*

Le mode None n'apportant aucune sécurité, tous les objectifs sont logiquement attaquables. Dans le cas où les nonces échangés en mode Sign ne sont pas chiffrés (absence de *KeyWrapping*), les clés sont accessibles à l'attaquant. Par ailleurs, lorsque l'identité du destinataire n'est pas incluse dans les signatures, l'intrus est en mesure de rediriger les messages d'une session à l'autre afin de mettre en défaut les propriétés d'authentification par les nonces [17]. Cette attaque est présentée en figure 4.6 et, compromet l'authentification de C avec N_C en mode SignEncrypt.

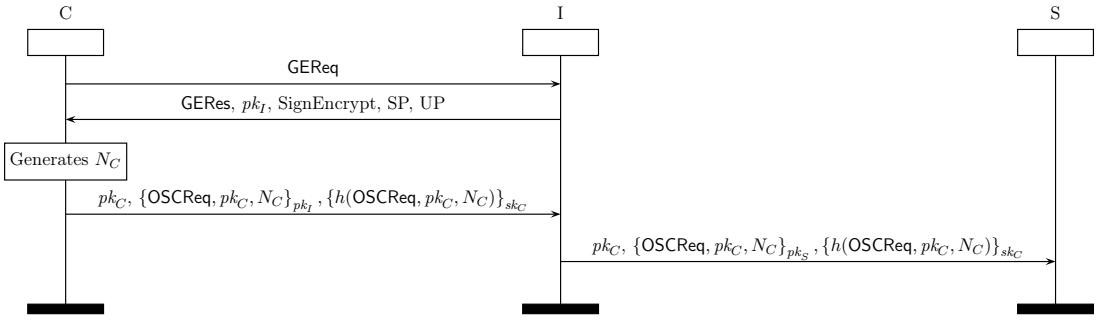


FIGURE 4.6. – Attaque sur N_C : I usurpe C en parlant à S

Nous testons dans un second temps le protocole avec les nonces chiffrés en mode Sign (utilisation de *KeyWrapping*) et où l'identité des destinataires est incluse dans les signatures. Ainsi, la clé publique du destinataire de chaque message est incluse dans la

16. Traduction de « If a thumbprint is not specified this value may be 0 or -1. »
17. Dans le cas où il s'agit d'une redirection et non d'un rejeu, cette attaque ne sera pas affectée par les estampillages (*timestamps*).

signature et toutes les occurrences de N_C sont remplacées par $\{N_C\}_{pk_S}$ dans le message 3 (resp. pour N_S dans le message 4). La totalité du message est signée mais seuls les nonces sont chiffrés. Plus formellement, les messages 3 et 4 de la figure 4.5 sont remplacés par :

3. $C \to S$: $\mathsf{OSCReq}, pk_C, \{N_C\}_{pk_S}, pk_S, \left\{h(\mathsf{OSCReq}, pk_C, \{N_C\}_{pk_S}, pk_S)\right\}_{sk_C}$
4. $S \to C$: $\mathsf{OSCRes}, \{N_S\}_{pk_C}, ST, TTL, pk_C, \left\{h(\mathsf{OSCRes}, \{N_S\}_{pk_C}, ST, TTL, pk_C)\right\}_{sk_S}$

Il convient de préciser que ces contre-mesures sont classiques et étaient déjà proposées par Abadi et Needham en 1996 [Abadi and Needham, 1996]. Les résultats après correction sont présentés en table 4.3 où ✓ signifie que la propriété est vérifiée et ✗ qu'une attaque a été trouvée. Les résultats obtenus sur le protocole avec les contre-mesures montrent cette fois l'absence d'attaque sur les modes Sign et SignEncrypt. Nous avons par ailleurs testé les contre-mesures séparément et nous pouvons confirmer que les deux sont indépendantes.

Mode de sécurité OPC-UA	**Objectifs**			
	Sec K_{CS}	Sec K_{SC}	Auth N_S	Auth N_C
None	✗	✗	✗	✗
Sign	✓	✓	✓	✓
SignEncrypt	✓	✓	✓	✓

TABLE 4.3. – Résultats pour le sous-protocole *OpenSecureChannel* avec contre-mesures

4.3.3. OPC-UA *CreateSession*

Le sous-protocole *CreateSession* constitue la seconde partie du handshake d'OPC-UA. Il permet à un client d'envoyer ses identifiants (par exemple un login et un mot de passe) afin d'authentifier l'humain le contrôlant auprès du serveur via le canal établi par le sous-protocole *OpenSecureChannel* exécuté précédemment. Il conserve également le mode de sécurité qui a été établi lors de la création du canal et utilise les clés symétriques qui ont été dérivées. Ainsi, le chiffrement utilise des primitives symétriques et les signatures des *Message Authentication Codes* (MAC). Les messages envoyés par C sont chiffrés avec K_{CS} (resp. signés avec $KSig_{CS}$) et les messages envoyés par S sont chiffrés avec K_{SC} (resp. signés avec $KSig_{SC}$). Le sous-protocole *CreateSession* est présenté en figure 4.7.

Ce sous-protocole étant similaire à celui étudié en section 4.3.2, nous n'entrons pas dans le détail de chaque message. Le point clé de ce sous-protocole intervient dans le message 3 où le client envoie ses identifiants au serveur.

4.3.3.1. Modélisation avec ProVerif

Le sous-protocole *CreateSession* introduit donc la problématique de devoir modéliser des identifiants sous forme de login et mots de passe. Bien que cela ne présente aucune difficulté technique, ce n'est pas une tâche courante en vérification de protocoles cryptographiques. Pour cause, cela suppose de faire des hypothèses de modélisation en fonction de l'utilisation qui serait faite des mots de passe, par exemple le fait qu'un mot

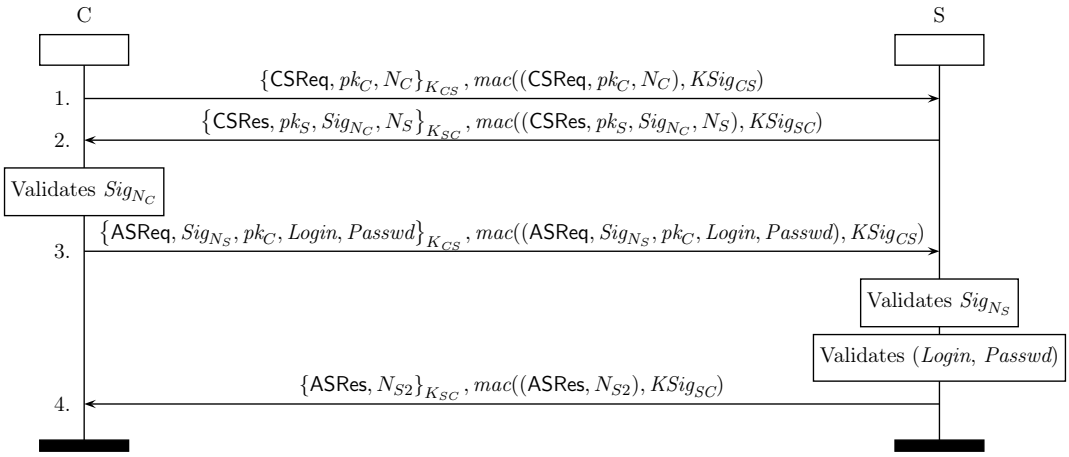


FIGURE 4.7. – Le sous-protocole OPC-UA *CreateSession* en mode SignEncrypt

de passe puisse être réutilisé plusieurs fois ou non. Dans notre cas, on peut par exemple considérer trois politiques d'utilisation des mots de passe possibles : (i) le client utilise toujours le même mot de passe, (ii) le client utilise un mot de passe différent pour chaque serveur, et (iii) le client utilise un mot de passe différent pour chaque connexion à un serveur (donc des mots de passe à usage unique ou *One-time password*). Or, pour rappel, nous considérons un attaquant dont les certificats seraient acceptés par les clients et les serveurs (il peut agir comme un client ou un serveur légitime). Ainsi, si le client utilise toujours le même mot de passe alors il communiquera volontairement son mot de passe à l'intrus en jouant le protocole avec lui. Aussi nous considérons que le client utilise un mot de passe différent pour chaque serveur.

Afin de modéliser des identifiants nous devons introduire des théories équationnelles permettant de relier un login à un mot de passe. Qui plus est, comme nous ajoutons l'hypothèse que chaque mot de passe n'est utilisé que pour un serveur, il faut aussi pouvoir modéliser le lien entre les identifiants et un serveur. Pour cela nous proposons deux fonctions : Login and Passwd. Le constructeur Login représente une valeur publique associée à un agent. Il prend pour cela en paramètre sa clé publique. Par exemple le login de C est le terme $Login(pk_C)$. Ce constructeur est public à tout le monde. L'intrus est donc en mesure de générer ses propres logins et de connaître le login du client (qui est public). La fonction Passwd quand à elle modélise un terme secret pour tout le monde à l'exception du possesseur du mot de passe. Elle prend pour cela la clé privée (secrète) associée à un agent. Afin de modéliser le lien entre mot de passe et serveur, elle prend aussi la clé publique du serveur en paramètre. Ainsi le mot de passe de C pour le serveur S sera $Passwd(sk_C, pk_S)$. Nous proposons enfin l'équation suivante pour qu'un serveur S puisse vérifier qu'un couple login/mot de passe d'un client C est valide est associé au bon serveur :

$$verifyCreds(sk_S, Login(pk_C), Passwd(sk_C, pk_S)) = true$$

Plusieurs hypothèses de modélisation découlent par ailleurs des résultats obtenus sur le sous-protocole précédent. L'intrus peut jouer légitimement le sous-protocole *OpenSecureChannel* avec n'importe quel participant. Nous faisons donc l'hypothèse qu'il peut donc peut partager une clé symétrique avec les autres participants. Cependant, comme expliqué en section 4.3.2.2, l'intrus n'est pas en mesure d'obtenir la clé symétrique d'une session où il a usurpé l'identité d'un autre participant. Cela signifie que si l'intrus partage une clé symétrique avec un participant, ce dernier est conscient de parler avec l'intrus.

Enfin, nous avons identifié précédemment que la confidentialité des clé symétriques en mode Sign dépend du fait que les nonces utilisés soient chiffrés ou non. Toujours pour pallier ces manques de clarté de la documentation sur le sujet, nous vérifions le sous-protocole *CreateSession* en mode Sign dans le cas où l'intrus a accès à des clés symétriques partagées entre le client et le serveur et dans le cas contraire. La question de chiffrer ou non les nonces en mode Sign s'applique aussi aux mots de passe échangés. En l'absence de chiffrement, nous considérons que le mot de passe est envoyé en clair et vérifions également les deux possibilités. Dans ce contexte, nous considérons quatre objectifs : (i) le secret du mot de passe, (ii) l'authentification de C par son mot de passe, (iii) l'authentification de C par Sig_{N_S}, et (iv) l'authentification de S par Sig_{N_C}.

4.3.3.2. Résultats

Les résultats sans *KeyWrapping* (c'est à dire avec les clés connues de l'attaquant en mode Sign et les mots de passe envoyés en clair) sont présentés en table 4.4. Le symbole ✓ signifie que la propriété est vérifiée et ✗ qu'une attaque a été trouvée. À nouveau, tous les objectifs sont logiquement attaquables en mode None. De plus, le secret du mot de passe est compromis puisque celui-ci est envoyé en clair par le client. Cependant, l'authentification par les nonces est tout de même préservée puisqu'ils sont signés avec les clés secrètes des participants et non les clé symétriques. Une attaque est trouvée sur l'authentification par le mot de passe en mode Sign. L'intrus y remplace les identifiants par d'autres identifiants envoyés en clair et recalcule le MAC avec les clés symétriques corrompues. Cette attaque signifie que l'intrus est en mesure de voler la session de C juste après que celui-ci ait répondu au challenge de S.

Mode de sécurité OPC-UA	**Objectifs**			
	Sec *Passwd*	Auth *Passwd*	Auth Sig_{N_S}	Auth Sig_{N_C}
None	✗	✗	✗	✗
Sign	✗	✗	✓	✓
SignEncrypt	✓	✓	✓	✓

TABLE 4.4. – Résultats pour le sous-protocole *CreateSession*

Dans le cas où les nonces échangés pendant le sous-protocole *OpenSecureChannel* son chiffrés (c'est à dire avec les clés symétriques secrètes et le mot de passe de C envoyé chiffré), le secret du mot de passe et son authentification sont alors vérifiés. Ces résultats sont présentés en table 4.5 où ✓ signifie que la propriété est vérifiée et ✗ qu'une attaque a été trouvée. A nouveau, nous avons vérifié l'indépendance des contre-mesures. Cela montre à quel point il est crucial de chiffrer les nonces et les mots de passe en mode Sign et les cas contraires devraient être très clairement explicités dans la documentation.

Mode de sécurité OPC-UA	**Objectifs**			
	Sec *Passwd*	Auth *Passwd*	Auth Sig_{N_S}	Auth Sig_{N_C}
None	✗	✗	✗	✗
Sign	✓	✓	✓	✓
SignEncrypt	✓	✓	✓	✓

TABLE 4.5. – Résultats pour le sous-protocole *CreateSession* avec contre-mesures

Cette étude constitue à notre connaissance la première vérification formelle du handshake OPC-UA. Nous avons analysé les sous-protocoles avec et sans plusieurs fonctionnalités de sécurité qui s'avèrent toutes nécessaires. La documentation devrait donc être claire sur les points traités.

4.4. Nouvelle classe de propriétés adaptées aux systèmes industriels

Comme expliqué en section 1.1.2, l'intégrité est une propriété fondamentale à garantir pour les systèmes industriels. Cette notion peut beaucoup varier suivant le contexte et les communautés qui l'emploient. Dans cette section, nous étudions l'intégrité de messages échangés sur un réseau potentiellement non sécurisé. Traditionnellement, l'intégrité d'un message signifie le protéger contre des modifications accidentelles à l'aide de code de détection ou de corrections d'erreurs tels que des *Cyclic Redundancy Checks* ou CRC. Cependant, ces protections sont inefficaces contre des attaquants car ceux-ci peuvent aisément modifier le message et recalculer le CRC. De même, le protocole TCP protège contre les modifications accidentelles de l'ordre des messages à l'aide des numéros de séquence et d'acquittement mais ces protections sont tout aussi inefficaces contre un attaquant qui peut le modifier également. Ainsi, pour protéger les messages face à des intrus actifs, l'utilisation de primitives cryptographiques comme des *Message Authentication Codes* ou MAC est requise.

Cependant, ces propriétés ne sont que très peu étudiées en pratique par les outils de vérification de protocoles cryptographiques qui se focalisent principalement sur le secret et l'authentification. De façon similaire à la définition de l'authentification en section 4.2.2, plusieurs notions d'intégrité peuvent être considérées. Une première notion est le fait qu'un message ne soit pas modifié pendant son transfert. Cependant, des notions beaucoup plus larges d'intégrité peuvent être considérées, comme l'ordre des messages qui est important pour la plus part des systèmes d'information mais véritablement crucial pour les systèmes industriels. Nous proposons une formalisation de différentes notions de propriétés d'intégrité, actuellement inexistantes dans les outils de vérification. Nous appelons cette classe de propriété « intégrité du flux ». Dans l'article [Dreier et al., 2017b], nous implémentons ces propriétés dans l'outil Tamarin. Cette implémentation avec Tamarin n'étant pas une contribution directe de cette thèse, nous renvoyons le lecteur vers l'article. La section 4.4.1 présente les propriétés d'intégrité du flux et explique comment ces propriétés sont liées les unes aux autres. Ensuite, la section 4.4.2 présente les résultats d'analyse de ces propriétés sur les protocoles MODBUS et OPC-UA.

4.4.1. Définition de l'intégrité du flux

Dans cette section, nous commençons par introduire section 4.4.1.1 des notations afin de définir les propriétés d'intégrité du flux. Nous définissons ensuite les propriétés en section 4.4.1.2. En section 4.4.1.3, nous expliquons les relations entre ces propriétés. Enfin, nous instancions ces propriétés sur quelques exemples simples en section 4.4.1.4.

4.4.1.1. Notations

Dans nos définitions, nous parlons de séquences de messages. Soit S^* l'ensemble des séquences d'un ensemble S. Pour une séquence s, on dénote s_i son i-ème élément, $|s|$ pour sa longueur, et $idx(s) = \{1, \ldots, |s|\}$ pour l'ensemble de ses indices. On note [] la séquence vide, $[s_1, \ldots, s_k]$ la séquence s de longueur k, et $s \cdot s'$ la concaténation des séquences s et s'. On dit que la séquence $[s_1 \ldots s_n]$ est une *sous-chaîne* de la séquence $[r_1 \ldots r_m]$ s'il existe[18] une séquence $z_0, \ldots, z_n$ telle que :

$$z_0 \cdot [s_1] \cdot z_1 \cdot [s_2] \cdot \ldots \cdot [s_{k-1}] \cdot z_{n-1} \cdot [s_n] \cdot z_n = [r_1 \ldots r_m]$$

On note $set(S)$ l'ensemble non ordonné qui contient une seule fois chaque élément de la séquence S, et $mset(S)$ le *multiset* non ordonné qui contient les éléments de S. Pour distinguer les opérations sur les multisets des opérations sur les ensembles on utilise la notation $^{\#}$: par exemple $\cup$ dénote l'union d'ensembles et $\cup^{\#}$ l'union de multiset. Enfin, on utilise la notation $\{\cdot\}$ pour les ensembles et les multisets lorsque la différence se déduit du contexte.

Comme expliqué en chapitre 2, les protocoles de handshake, qui sont étudiés en général par les outils de vérification de protocoles cryptographiques, sont des séquences finies de messages. Nous étudions en revanche des protocoles de transport qui visent à transporter des commandes ou des données d'un agent à l'autre, ces séquences de messages pouvant être potentiellement infinies. On appelle les commandes transportées le *payload* (i.e. : la charge) du message par opposition notamment aux en-têtes. Ainsi, pour identifier le payload dans un message, nous utilisons des types. Le payload a ainsi pour type D pour données, et le reste du message aura potentiellement un autre type (par exemple H pour un haché ou S pour une signature). Un exemple est donné dans le protocole 1 en section 4.4.1.4.

4.4.1.2. Définitions des propriétés

Les messages peuvent être échangés sur le réseau par un nombre arbitraire d'agents en même temps. On dénote *Flow Integrity* (intégrité du flux) pour le flux de messages entre deux agents A et B. Plus précisément, on définit l'intégrité des payloads. C'est à dire qu'on cherche à protéger le contenu applicatif des messages, et non les en-têtes, c'est à dire les sous-termes de type D (pour données) des messages. Nous faisons cette restriction car les protocoles de transport ont généralement tendance à ne protéger que les parties critiques des messages afin de limiter les calculs cryptographiques. Ainsi, si un protocole envoie un message avec sa signature, le tout accompagné d'un identifiant non critique (et donc non signé), ce protocole n'assurerait aucune notion d'intégrité,

18. z_i pouvant être la séquence vide.

l'identifiant non critique pouvant être modifié alors que le payload, lui, ne peut pas être modifié. Ainsi, se limiter à l'intégrité du payload nous permet d'éviter de fausses attaques.

Définition 2. *Soit $S_{A,B,D}$ la séquence qui contient les sous-termes de type D de tous les messages envoyés par l'agent A à l'agent B, et la séquence $R_{A,B,D}$ qui contient les sous-termes de type D de tous les messages reçus par l'agent B venant de A.*

Par exemple, lorsqu'un protocole envoie le message m de type D avec un haché $h(m)$ de type H, $S_{A,B,D}$ contient uniquement m, et non le haché comme expliqué dans le protocole 1 en section 4.4.1.4. Comme A peut envoyer des messages à B mais aussi à d'autres agents comme par exemple C, et B peut recevoir des messages de A et de E, on définit la séquence ordonnée des messages que A envoie à B, et la séquence ordonnée des messages que B reçoit de A. A noter que les notations d'origine et de destinations sont du point de vue des agents. C'est à dire qu'un message m de type D appartient à $R_{A,B,D}$ si B *pense* l'avoir reçu de A. De même, si un message m de type D que A *souhaite* envoyer à B, mais qu'il est reçu par C, on a m qui appartient à $S_{A,B,D}$.

Nous définissons maintenant trois niveaux d'intégrité, d'authenticité et de transmission où chaque niveau d'intégrité est défini comme la conjonction des propriétés d'authenticité et de transmission du niveau correspondant. Intuitivement, l'authenticité assure qu'un message n'a pas été altéré pendant l'envoi et la transmission assure que le message n'a pas été perdu. La première notion d'authenticité requiert que tous les messages reçus d'un agent aient bien été envoyés par celui-ci (ils peuvent cependant être perdus ou dupliqués).

Propriété 1. *Un protocole assure la propriété* Non-Injective Message Authenticity *ou NIMA (pour authenticité non injective des messages) entre un agent émetteur A et un agent destinataire B pour des messages de type D si* $\mathrm{set}(R_{A,B,D}) \subseteq \mathrm{set}(S_{A,B,D})$.

Cette propriété vérifie que les messages n'ont par été modifiés mais pas qu'ils ont bien été reçus. Pour cela, nous définissons la propriété de transmission correspondante.

Propriété 2. *Un protocole assure la propriété* Non-Injective Message Delivery *ou NIMD (pour transmission non injective des messages) entre un agent émetteur A et un agent destinataire B pour des messages de type D si* $\mathrm{set}(R_{A,B,D}) \supseteq \mathrm{set}(S_{A,B,D})$.

Il convient de noter qu'assurer la transmission des messages est compliqué dans un réseau asynchrone non sécurisé comme Internet. Cependant, les systèmes industriels utilisent souvent des protocoles particuliers ajoutant de la redondance et offrant ainsi des garanties de transmission. On peut par exemple citer les protocoles *Parallel Redundancy Protocol (PRP)* et *High-availability Seamless Redundancy (HSR)* [IEC-62439, 2016]. Si l'on combine les deux propriétés précédentes, on obtient l'intégrité non injective des messages.

Propriété 3. *Un protocole assure la propriété* Non-Injective Message Integrity *ou NIMI (pour intégrité non injective des messages) entre un agent émetteur A et un agent destinataire B pour des messages de type D si* $\mathrm{set}(R_{A,B,D}) = \mathrm{set}(S_{A,B,D})$.

On peut aisément remarquer qu'un protocole qui assure NIMI assure également NIMA et NIMI, et vice versa par propriété de l'inclusion d'ensembles. Pour assurer que les messages ne puissent pas être dupliqués, nous proposons les propriétés *Injective Message Authenticity* et *Injective Message Delivery*. Ces propriétés sont l'équivalent des précédentes où les ensembles sont remplacés par des multisets afin de tenir compte du nombre d'occurrences de chaque message.

Propriété 4. *Un protocole assure la propriété* Injective Message Authenticity *ou IMA (pour authenticité injective des messages) entre un agent émetteur A et un agent destinataire B pour des messages de type D si* $\text{mset}(R_{A,B,D}) \subseteq^{\#} \text{mset}(S_{A,B,D})$.

Propriété 5. *Un protocole assure la propriété* Injective Message Delivery *ou IMD (pour transmission injective des messages) entre un agent émetteur A et un agent destinataire B pour des messages de type D si* $\text{mset}(R_{A,B,D}) \supseteq^{\#} \text{mset}(S_{A,B,D})$.

En vérifiant les deux propriétés à la fois, on garantit la propriété *Injective Message Integrity* ou IMI (pour intégrité injective des messages).

Propriété 6. *Un protocole assure la propriété* Injective Message Integrity *ou IMI (pour intégrité injective des messages) entre un agent émetteur A et un agent destinataire B pour des messages de type D si* $\text{mset}(R_{A,B,D}) =^{\#} \text{mset}(S_{A,B,D})$.

A nouveau, on peut aisément remarquer qu'un protocole qui assure IMI assure également IMA et IMI, et vice versa par propriété de l'inclusion des multisets. IMI assure que les messages seront préservés, transmis, non dupliqués mais peuvent encore être réordonnées. On introduit pour cela les propriétés de *Flow Authenticity* et *Flow Delivery*.

Propriété 7. *Un protocole assure la propriété* Flow Authenticity *ou FA (pour authenticité du flux) entre un agent émetteur A et un agent destinataire B pour des messages de type D si $R_{A,B,D}$ est une sous-chaîne $S_{A,B,D}$.*

Propriété 8. *Un protocole assure la propriété* Flow Delivery *ou FD (pour transmission du flux) entre un agent émetteur A et un agent destinataire B pour des messages de type D si $S_{A,B,D}$ est une sous-chaîne $R_{A,B,D}$.*

En vérifiant les deux propriétés à la fois, on garantit la propriété *Flow Integrity* ou FI (pour intégrité du flux).

Propriété 9. *Un protocole assure la propriété* Flow Integrity *ou FI (pour intégrité du flux) entre un agent émetteur A et un agent destinataire B pour des messages de type D si $S_{A,B,D} = R_{A,B,D}$.*

A nouveau, on peut aisément remarquer qu'un protocole qui assure IMI assure également IMA et IMI, et vice versa par propriété de l'inclusion de chaînes.

Canaux résilients : On suppose que les agents communiquent sur un réseau contrôlé par un intrus Dolev-Yao. Celui-ci peut effectuer diverses actions sur les messages transitant sur le réseau, à commencer par les bloquer. On se doute intuitivement que dès lors aucune propriété de transmission ne pourra être vérifiée si les messages peuvent ne

pas être reçus. Aussi, afin de pouvoir vérifier ces propriétés et prendre en compte les protocoles de redondances comme PRP et HSR, nous introduisons la notion de canal résilient. Un canal est dit résilient si les messages envoyés finissent forcément par arriver. L'intrus contrôlant le canal peut cependant les retarder durant un temps fini, modifier leur expéditeur ou les rediriger vers un mauvais destinataire.

4.4.1.3. Relations entre les propriétés

Un protocole qui assure la propriété FI assure également la propriété IMI, et de même un protocole assurant IMI assure également NIMI. La même remarque s'applique de façon analogue pour les propriétés d'authenticité et de transmission. Ces relations sont résumées dans la figure 4.8.

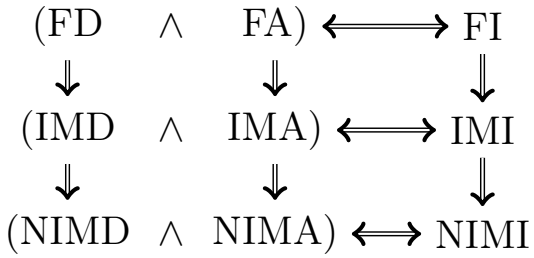


FIGURE 4.8. – Relations entre nos propriétés : $A \Rightarrow B$ est vraie si un protocole assurant A assure aussi B.

De plus, un protocole qui assure soit FA et IMD, soit FD et IMA, assure également FI, comme expliqué en théorème 6.

Théorème 6. *Un protocole qui assure FD et IMA assure également FI (FD ∧ IMA ⇒ FI). De façon analogue, un protocole qui assure FA and IMD, assure également FI (FA ∧ IMD ⇒ FI).*

Démonstration. Soit $[s_1, \ldots, s_n] = S_{A,B,D}$ et $[r_1, \ldots, r_m] = R_{A,B,D}$. Supposons qu'un protocole assure FD et IMA, i.e. : on a $S_{A,B,D}$ une sous-chaîne de $R_{A,B,D}$, et $mset(R_{A,B,D}) \subseteq mset(S_{A,B,D})$. De plus, comme tout protocole assurant FD assure également IMD, on a $mset(R_{A,B,D}) \supseteq mset(S_{A,B,D})$, et donc $mset(R_{A,B,D}) = mset(S_{A,B,D})$. Cela signifie que $n = m$, i.e. : les deux séquences ont la même longueur. Par définition des sous-chaînes, il existe des sous-séquences $z_0, \ldots, z_n$ telles que $z_0 \cdot [s_1] \cdot z_1 \cdot \ldots \cdot z_{n-1} \cdot [s_n] \cdot z_n = [r_1 \ldots r_m]$. Comme $n = m$, on a $[s_1, \ldots, s_n] = [r_1, \ldots, r_n]$, qui est ce que nous voulions montrer. La seconde preuve est analogue. □

4.4.1.4. Exemples simples

Nous présentons deux exemples simples afin d'illustrer les propriétés d'authenticité, de transmission et d'intégrité du flux.

Protocole 1. *Soit le protocole suivant où* Alice *(A) envoie à* Bob *(B) un message m ainsi que son haché :*

$$A \to B : m, h(m)$$

Le payload du message est m qui a pour type D (pour données), tandis que $h(m)$ est de type H (pour haché). Ainsi, si A envoie le message $(m_1, h(m_1))$ à B (et qui est réellement reçu par B), alors on a $S_{A,B,D} = R_{A,B,D} = [m_1]$, comme seuls les sous-termes de type D sont considérés. On remarque facilement que ce protocole n'assure aucune des propriétés d'authenticité définies ci-dessus puisqu'un attaquant peut aisément recalculer le haché d'un message après l'avoir modifié. Même en supposant un canal résilient où les messages serait toujours reçus, ce protocole n'assure aucune propriété de transmission car les messages peuvent être redirigés vers un mauvais destinataire qui ne pourra pas le remarquer.

Protocole 2. *Soit le protocole suivant où* Alice *(A) envoie à* Bob *(B) un message m accompagné d'un MAC utilisant une clé symétrique K_{AB} partagée entre A et B :*

$$A \to B : m, \mathrm{mac}(m, K_{\mathrm{AB}})$$

Sur un réseau non sécurisé, ce protocole assure la propriété NIMA car B est en mesure de vérifier le MAC et peut s'assurer que le message vient donc bien de A et lui est destiné. Cependant les messages peuvent encore être dupliqués et réordonnés. De plus, il requiert un canal résilient pour assurer la propriété NIMD.

Afin d'éviter que des messages puissent être dupliqués et assurer leur ordre, on peut utiliser des estampillages ou des compteurs (par exemple des numéros de séquence) comme le montreront les études de cas en section 4.4.2.

4.4.2. Application à des protocoles de communication industriels

Nous examinons la sécurité de plusieurs variantes de deux protocoles de communication industriels (à savoir MODBUS et OPC-UA) pour vérifier s'ils garantissent les propriété d'intégrité du flux que nous avons défini en section 4.4.1. Dans le cadre de l'article [Dreier et al., 2017b], ces propriétés ont été implémentées dans l'outil Tamarin en collaboration avec des développeurs de l'outil [19]. Nous nous focalisons ici sur les protocoles de transport décrits en section 2.2. À la différence des protocoles de handshake qui visent à établir des sessions, les protocoles de transport sont utilisés une fois la session établie pour transporter les commandes envoyées par le client au serveur. Nous considérons un nombre infini de sessions en parallèle, chacune pouvant être constituée d'un nombre de commandes arbitrairement grand.

4.4.2.1. Analyse de MODBUS

Comme décrit en section 2.2.3.1, MODBUS est un protocole de communication industriel crée en 1979 par Modicon (maintenant Schneider Electric). C'est l'un des protocoles les plus populaires du domaine et peut être utilisé aussi bien avec des bus série que via des communications TCP (nous nous focalisons ici sur la version TCP). Seul le client peut envoyer des requêtes (le serveur n'envoie jamais de messages de lui-même). La version TCP inclut un numéro de séquence applicatif en plus de celui fourni par TCP. Ce

19. Les modèles utilisés peuvent être trouvés à l'adresse suivante : https://maxime.puys.name/files/phdThesisAssets.tar.bz2

numéro est appelé *transaction identifier* et incrémente de un à chaque requête du client. D'autres termes applicatifs sont inclus dans le message (le champ *protocol identifier* présent par compatibilité avec la version série, la longueur du message applicatif et le *unit identifier* utilisé pour dispatcher les commandes aux capteurs et actionneurs). Ces trois termes n'impactant pas la sécurité, nous les modélisons comme un unique en-tête public *ph*. La figure 4.9 montre deux requêtes et réponses MODBUS avec n le *transaction identifier* et *ph* l'en-tête public.

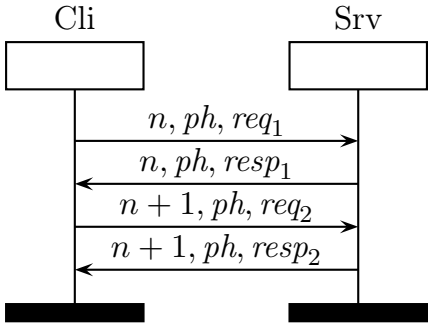


FIGURE 4.9. – Deux requêtes et réponses MODBUS

Ce protocole s'appuie sur TCP pour se protéger des erreurs de transmission via des sommes de contrôle mais n'implémente pas de contre-mesures contre des attaquants malicieux. Ainsi, n'importe qui peut modifier le contenu d'un message et recalculer les sommes de contrôle l'accompagnant, rendant possible l'envoi de commandes arbitraires. Afin d'éviter cela, deux variantes de MODBUS ont été proposées dans des travaux académiques. En 2009, Fovino *et al.* [Fovino et al., 2009], ont proposé une version ajoutant du chiffrement, des signatures cryptographiques et l'estampillage des messages. La figure 4.10 présente la même session qu'en figure 4.9 avec les contre-mesures de [Fovino et al., 2009] où ts_i représente l'estampillage du i-ème message.

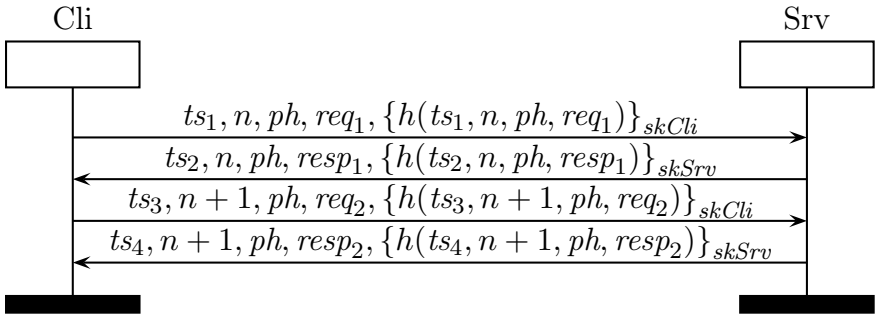


FIGURE 4.10. – Deux requêtes et réponses MODBUS de [Fovino et al., 2009]

En 2013, Hayes et El-Khatib [Hayes and El-Khatib, 2013] proposent une autre version ajoutant également des signatures cryptographiques mais se basant sur le protocole de transmission SCTP (*Stream Control Transmission Protocol*) au lieu de TCP. Comme TCP, SCTP ne protège pas des modifications malicieuses des messages. A la différence

de Fovino *et al.*, Hayes et El-Khatib utilisent des primitives symétriques, ici des MACs. De plus, pour éviter que les messages puissent être rejoués, ils incluent dans les MACs un nonce fourni par SCTP, appelé *verification tag*. La figure 4.11 présente la même session qu'en figure 4.9 avec les contre-mesures de [Hayes and El-Khatib, 2013] où *vt* représente le *verification tag* de la session SCTP.

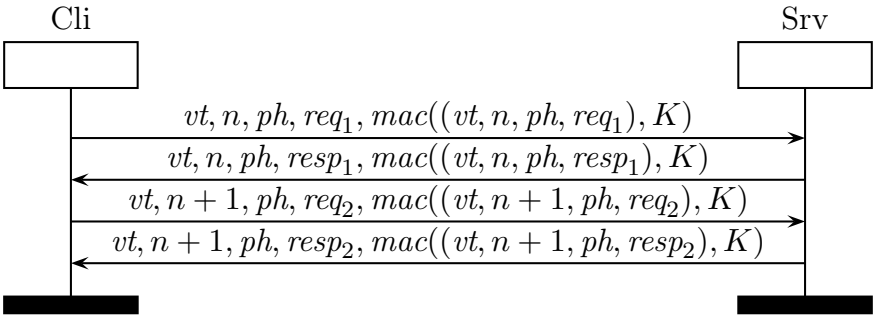


FIGURE 4.11. – Deux requêtes et réponses MODBUS de [Hayes and El-Khatib, 2013]

Résultats de l'analyse : Nous avons modélisé les trois versions de MODBUS décrites ci-dessus avec Tamarin. Nous analysons d'abord les protocoles sur un canal classique non résilient. Les résultats sont présentés en table 4.6 où ✓ signifie que la propriété est vérifiée et ✗ qu'une attaque a été trouvée.

Protocole	NIMA	IMA	FA	NIMD	IMD	FD
Standard MODBUS [MODBUS, 2004]	✗	✗	✗	✗	✗	✗
MODBUS Sign [Fovino et al., 2009]	✗	✗	✗	✗	✗	✗
MODBUS MAC [Hayes and El-Khatib, 2013]	✓	✓	✓	✗	✗	✗

TABLE 4.6. – Résultats pour MODBUS sur un canal classique

Tamarin trouve des attaques sur toutes les propriétés pour la version standard. C'est normal étant donné qu'elle ne propose aucune fonctionnalité de sécurité. En revanche, on peut remarquer que la version de [Fovino et al., 2009] utilisant des clés asymétriques est également vulnérable. Cela s'explique par le fait que l'identité du destinataire n'est pas présente dans la signature. Un attaquant est donc en mesure de rediriger un message vers un mauvais destinataire, violant ainsi toutes nos propriétés. On remarquera que la même attaque était présente dans l'analyse du handshake OPC-UA en section 4.3. En revanche, la version proposée par [Hayes and El-Khatib, 2013] est sûre car les clés cryptographiques utilisées sont symétriques. Ainsi, sous l'hypothèse qu'elles sont propres à deux participants, un mauvais destinataire ne pourra pas vérifier le MAC accompagnant le message et s'apercevra d'un problème. Cette version assure donc toutes nos propriétés d'authenticité mais ne vérifie pas les propriétés de transmission puisque l'intrus peut simplement supprimer les messages. Si l'on réalise la même analyse avec un canal résilient, cette version vérifie alors toutes les propriétés de transmission comme présenté en table 4.7 où ✓ signifie que la propriété est vérifiée et ✗ qu'une attaque a été trouvée.

Les autres versions du protocole échouent toujours à les vérifier puisqu'il reste possible de rediriger un message vers un mauvais destinataire comme expliqué ci-dessus.

Protocole	NIMA	IMA	FA	NIMD	IMD	FD
Standard MODBUS [MODBUS, 2004]	✗	✗	✗	✗	✗	✗
MODBUS Sign [Fovino et al., 2009]	✗	✗	✗	✗	✗	✗
MODBUS MAC [Hayes and El-Khatib, 2013]	✓	✓	✓	✓	✓	✓

TABLE 4.7. – Résultats pour MODBUS sur un canal résilient

4.4.2.2. Analyse de OPC-UA

Comme présenté en chapitre 2, OPC-UA est un des protocoles de communication industriels les plus récents. Publié en 2006 puis standardisé en 2015 [IEC-62541, 2015], il a été développé par la Fondation OPC. Il est souvent considéré par les experts du domaine comme le futur standard des communications industrielles, de part la présence des principaux acteurs dans la fondation l'ayant développé. Après avoir analysé ses protocoles de handshake avec ProVerif en section 4.3, nous nous intéressons maintenant à son protocole de transport [20]. Ce protocole utilise les clés générées durant le handshake et suit les modes de sécurité négociés pendant celui-ci, à savoir :

— *None* qui n'apporte aucune sécurité (par compatibilité avec les matériels trop anciens ou ayant une trop faible puissance pour faire de la cryptographie) ;

— *Sign* où les messages sont signés mais envoyés en clair ;

— *SignEncrypt* où les messages sont signés et chiffrés.

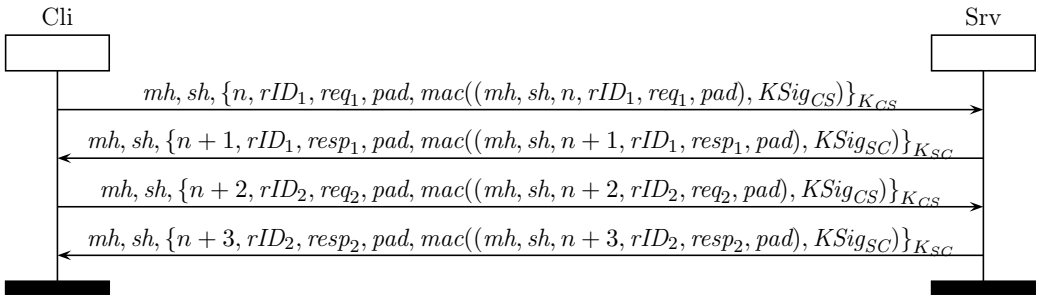


FIGURE 4.12. – Deux requêtes et réponses OPC-UA en mode SignEncrypt

La figure 4.12 montre deux requêtes et réponses OPC-UA avec :

— *mh* pour *message header* contenant des termes publics ;

— *sh* pour *security header* représentant un nonce frais appelé *security token* (similaire au nonce offert par SCTP) ;

— n est un numéro de séquence s'incrémentant à chaque requête *et* réponse ;

— rID_i représente l'identifiant d'une requête afin que les réponses ne soient pas mélangées entre elles ;

20. https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-6-mappings/ Version 1.03 du 20/07/2015.

— req_i (resp. $resp_i$) est le contenu applicatif de la requête (resp. réponse) ;

— pad est un padding ajouté si nécessaire ;

— $mac(\cdot,\cdot)$ est le MAC de tous les termes ci-dessus.

Seul le numéro de séquence, le contenu du message et les signatures sont chiffrés. Ainsi, suivant le mode de sécurité :

— En mode Sign, le message 1 (resp. 2, 3, et 4) de la figure 4.12 devient :

$$mh, sh, n, rID_1, req_1, pad, mac((mh, sh, n, rID_1, req_1, pad), KSig_{CS})$$

— En mode None, le message 1 (resp. 2, 3, et 4) de la figure 4.12 devient :

$$mh, sh, n, rID_1, req_1, pad$$

Résultats de l'analyse : Nous avons modélisé les trois modes de sécurité d'OPC-UA avec Tamarin. Nous analysons d'abord les protocoles sur un canal classique non résilient. Les résultats sont présentés en table 4.8 où ✓ signifie que la propriété est vérifiée et ✗ qu'une attaque a été trouvée.

Protocole	NIMA	IMA	FA	NIMD	IMD	FD
OPC-UA None	✗	✗	✗	✗	✗	✗
OPC-UA Sign	✓	✓	✓	✗	✗	✗
OPC-UA SignEncrypt	✓	✓	✓	✗	✗	✗

TABLE 4.8. – Résultats pour OPC-UA sur un canal classique

Tamarin trouve des attaques sur les propriétés du mode None. À nouveau, ce n'est pas surprenant puisque cette version n'apporte aucune fonctionnalité de sécurité. Par contre, les versions Sign et SignEncrypt garantissent toutes les propriétés d'authenticité. Cela signifie que les MACs ajoutés par le mode Sign sont suffisants pour garantir la propriété FA (ainsi que NIMA et IMA par extension). On retrouve d'ailleurs cette conclusion dans la version de MODBUS proposée dans [Hayes and El-Khatib, 2013] qui introduit des MACs mais pas de chiffrement. Par curiosité, nous avons étudié une variante du protocole avec uniquement du chiffrement symétrique et pas de MAC (c'est à dire une version du protocole n'existant pas en vrai). Il apparaît que nous obtenons les mêmes résultats que dans le mode Sign. Ce n'est donc pas les MACs en eux-mêmes qui importent mais le fait que les clés ne soient partagées qu'entre deux individus. Pour pouvoir garantir les propriétés de transmission, nous réalisons l'analyse avec un canal résilient, présentée en table 4.9 où ✓ signifie que la propriété est vérifiée et ✗ qu'une attaque a été trouvée.

Protocole	NIMA	IMA	FA	NIMD	IMD	FD
OPC-UA None	✗	✗	✗	✗	✗	✗
OPC-UA Sign	✓	✓	✓	✓	✓	✓
OPC-UA SignEncrypt	✓	✓	✓	✓	✓	✓

TABLE 4.9. – Résultats pour OPC-UA sur un canal résilient

OPC-UA dans le cas de compteurs bornés : Les compteurs que nous avons modélisés jusqu'à présent pour représenter les numéros de séquences étaient constitués d'entiers non bornés (appartenant donc à $\mathbb{N}$). Cependant les entiers machine sont bien entendu bornés et cela peut avoir un impact sur les propriétés liées à ces compteurs telles que l'intégrité du flux. Ainsi pour évaluer les conséquences de cette précision, nous réalisons à nouveau l'analyse du protocole OPC-UA en mode SignEncrypt (décrit en figure 4.12) avec des numéros de séquence explicitement bornés. Pour limiter les calculs et faciliter la lecture des résultats, nous bornons les numéros de séquence à quatre valeurs possibles. Cela signifie que le cinquième message envoyé par le client dans une session aura le même numéro de séquence que le premier. Pour cela nous introduisons une restriction en Tamarin imposant qu'un numéro de séquence soit exclusivement $Z, S(Z), S(S(Z))$ ou $S(S(S(Z)))$. Les résultats obtenus par Tamarin sur cette version sont présentés en table 4.10 où ✓ signifie que la propriété est vérifiée et ✗ qu'une attaque a été trouvée. Il apparaît que la propriété FA n'est plus vérifiée.

Protocole	NIMA	IMA	FA	NIMD	IMD	FD
OPC-UA SignEncrypt avec compteurs bornés et canal classique	✓	✓	✗	✗	✗	✗
OPC-UA SignEncrypt avec compteurs bornés et canal résilient	✓	✓	✓	✓	✓	✓

TABLE 4.10. – Résultats pour OPC-UA avec des compteurs bornés

L'attaque est présentée en figure 4.13. Le client envoie quatre messages, le quatrième ayant donc le même numéro de séquence que le premier. L'intrus retarde les trois premiers messages de façon à ce que le premier reçu par le serveur soit le quatrième envoyé par le client. Le serveur ne se rend pas compte de la différence puisque ce message porte le numéro de séquence qu'aurait le premier message envoyé. L'intrus transmet ensuite le second message que le serveur accepte également puisque son numéro de séquence (1) suit celui du quatrième message (0). L'ordre des messages a donc été modifié.

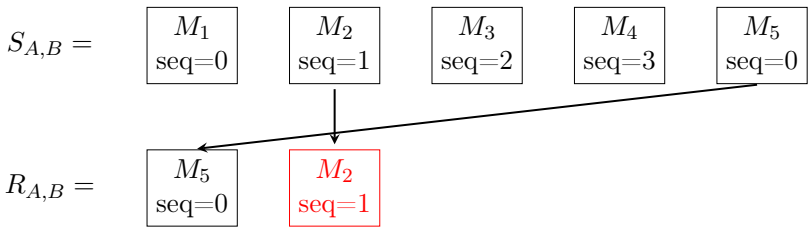


FIGURE 4.13. – Attaque contre FA sur OPC-UA avec des compteurs bornés

Il apparaît que l'attaque disparaît en cas de canal résilient. Cela est en fait dû à une incompatibilité entre la restriction modélisant les canaux résilients et celle modélisant les compteurs bornés. Cette incompatibilité a pour effet que le serveur n'accepte qu'une fois chaque numéro de séquence faisant disparaître *de facto* l'attaque. Cette analyse illustre le besoin que les compteurs soient exprimés sur un grand intervalle de

possibilités. De plus lorsque plus de messages doivent être envoyés que de numéros de séquences sont disponibles, la session doit être réinitialisée et les clés renégociées. C'est notamment la solution préconisée dans le standard OPC-UA (Partie 6, 6.7.2) : « Un numéro de séquence ne doit pas être réutilisé pour un même TokenId. La durée de vie du SecurityToken doit être suffisamment courte pour assurer que cela ne se produit pas [...] » [21]. On remarque par ailleurs que la contradiction entre les restrictions, menant à ne pas réutiliser les numéros de séquence côté client reproduit le comportement de cette contre-mesure et on peut donc confirmer qu'elle fait disparaître l'attaque. En revanche, à notre connaissance, ce problème n'est évoqué ni par [Fovino et al., 2009], ni par [Hayes and El-Khatib, 2013], qui proposent à la place dans leurs versions sécurisées de MODBUS, d'utiliser des estampillages pour limiter la durée de vie des messages. Il convient alors que cette durée de vie soit suffisamment courte pour ne pas permettre de rejeu et cela déporte le problème du rejeu sur la sécurité du protocole de gestion d'horloges. Or ces derniers sont actuellement peu robustes avec par exemple 78 vulnérabilités (CVE) rapportées sur NTP entre le 6 août 2004 et le 27 mars 2017 (dont 33 entre 2016 et 2017) [22].

4.5. Conclusion

Dans ce chapitre, nous avons proposé deux contributions dans le but d'adapter les techniques de vérification de protocoles cryptographiques aux protocoles de communication industriels. En section 4.3, nous avons réalisé une analyse de propriétés de secret et d'authentification sur les protocoles de handshake d'OPC-UA à l'aide de l'outil ProVerif en modélisant le protocole à partir du standard. Ces travaux ont donné lieu à une publication à la conférence SAFECOMP 2016 [Puys et al., 2016a]. Ensuite en section 4.4, nous avons introduit une formalisation de propriétés basées sur l'ordre des messages, actuellement inexistantes dans les outils de vérification. Ces travaux ont donné lieu à une publication à la conférence SECRYPT 2017 [Dreier et al., 2017b]. Nous avons appelé cette classe de propriétés *intégrité du flux* et avons vérifié si les protocoles MODBUS et OPC-UA respectent ces propriétés. Ces deux axes ont été partiellement soutenus par le projet CNRS PEPS SISC ASSI 2016.

Par ailleurs, ces vérifications portant sur des modélisations, nous avons effectué une analyse rapide du code source du handshake d'OPC-UA implémenté dans la librairie python-opcua [23], utilisée pour l'API du dispositif de filtrage décrit en chapitre 3. Nous pouvons ainsi assurer que cette implémentation embarque toutes les contre-mesures que nous avons prouvées nécessaires. De plus, en s'intéressant au standard OPC-UA, nous avons pu relever d'autres points qui devraient être repris. En effet, le standard recommande notamment que les implémentations utilisent la primitive de hachage SHA1. Celle-ci datant de 1995 et est maintenant jugée obsolète par de nombreuses agences gouvernementales et grandes entreprises. Depuis janvier 2017, Microsoft, Google et Mozilla ont tous les trois annoncés que leurs navigateurs web ne supporteraient plus les

21. Traduction de « A SequenceNumber may not be reused for any TokenId. The SecurityToken lifetime should be short enough to ensure that this never happens [...] »
22. https://www.cvedetails.com/vulnerability-list/vendor_id-2153/NTP.html
23. https://github.com/FreeOpcUa/python-opcua

certificats SHA1. À la suite de nos analyses, nous avons fait remonter nos résultats à la Fondation OPC par le biais de nos partenaires industriels avec des recommandations en conséquences. Dans la version du standard à jour au moment de l'écriture de ce manuscrit [24], il apparait que certains points que nous avons mentionnés ont été clarifiés.

4.5.1. Positionnement rapport à l'état de l'art

Comme pointé en introduction, il est compliqué de se positionner par rapport à l'état de l'art étant donné que la très grande majorité des travaux de vérification de protocoles de communication industriels reposent sur des vérifications informelles. En effet, sur les neuf travaux recensés en section 4.1.4 et rappelés en table 4.11, seul deux sont formels. De plus, l'un date de 2005 et l'autre de 2016, on ne constate donc pas véritablement une prise de conscience sur la nécessité de preuves formelles de protocoles industriels. Par ailleurs, les deux travaux formels (à savoir [Graham and Patel, 2004, Amoah, 2016]) se focalisent tous les deux sur DNP3 tandis que nous analysons MODBUS et OPC-UA. Dans le cas des propriété d'intégrité du flux, à notre connaissance, cette formalisation constitue les premiers travaux introduisant ce genre de propriétés dans les outils de vérification de protocoles cryptographiques.

Référence	Année	Protocoles étudiés	Type d'analyse
[Clarke et al., 2004]	2004	DNP3, ICCP	Informelle
[Dzung et al., 2005]	2005	OPC, MMS, IEC/CEI 61850 ICCP, Ethernet/IP	Informelle
[Graham and Patel, 2004]	2004	DNP3	Formelle (OFMC)
[IEC-62541, 2015]	2006	OPC-UA	Informelle
[Patel and Yu, 2007]	2007	DNP3	Informelle
[Fovino et al., 2009]	2009	MODBUS	Informelle
[Hayes and El-Khatib, 2013]	2013	MODBUS	Informelle
[Wanying et al., 2015]	2015	MODBUS, DNP3, OPC-UA	Informelle
[Amoah, 2016]	2016	DNP3	Formelle (Réseaux de Petri)
[Puys et al., 2016a]	2016	OPC-UA	Formelle (ProVerif)
[Dreier et al., 2017b]	2017	MODBUS, OPC-UA	Formelle (Tamarin)

TABLE 4.11. – État de l'art des vérifications de protocoles industriels

4.5.2. Perspectives

Les vérifications de propriétés de secret et d'authentification étant aujourd'hui rendues très simples par les outils de vérification de protocoles cryptographiques, d'autres protocoles devraient être analysés. Il nous parait en effet grave que des protocoles contrôlant des systèmes aussi critiques que des centrales nucléaires ou des barrages puissent n'avoir jamais été vérifiés formellement. De plus comme mentionné ci-dessus, il est également très important de vérifier les implémentations des protocoles afin de vérifier si celles-ci sont conformes aux conclusions des analyses. Enfin, d'autres propriétés que le secret et l'authentification peuvent être considérées. Par exemple, les outils de vérification de protocoles cryptographiques permettent dès à présent de vérifier des propriétés

24. https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-6-mappings/ Version 1.04.27 du 18/07/2017.

de non-répudiation [Klay and Vigneron, 2009], autre propriété très importante pour les systèmes industriels. Une autre classe de propriétés très étudiée récemment est l'équivalence observationnelle. Cette propriété permet de vérifier qu'un intrus n'est pas en mesure de distinguer deux exécutions d'un protocole. Elle pourrait par exemple être intéressante lorsque des données clients sont manipulées (comme par exemple dans le cas des compteurs électriques intelligents).

Enfin, un autre défi de la vérification de protocoles cryptographiques réside dans les modèles incluant plusieurs protocoles partageant potentiellement des clés. Le fait que deux protocoles soient sûrs indépendamment l'un de l'autre ne signifie pas que la combinaison des deux l'est également. Une question similaire peut se poser pour l'encapsulation de protocoles, utilisée par compatibilité dans les milieux industriels. Par exemple il peut être intéressant d'encapsuler du trafic MODBUS dans des paquets OPC-UA pour le sécuriser au lieu de remplacer les automates programmables. Cependant il n'est pas certain que la sécurité induite par OPC-UA suffise à garantir la sécurité des deux protocoles [Dreier et al., 2017a].

Chapitre 5

A^{2}SPICS : Recherche automatique de scénarios d'attaques applicatives

> Be careful about using the following code - I've only proven that it works, I haven't tested it.
>
> *(Donald Knuth, 1977)*

Résumé du chapitre

Pour luter contre les APT (*Advanced Persistent Threats*), il est essentiel d'étudier au préalable quelles sont les faiblesses d'un système. Cela permet aussi de déterminer quels sont les attaquants qui pourraient s'en prendre au système et quels objectifs peuvent-ils atteindre. En particulier dans le cas de l'attaque Maroochy Shire, les employés on mis plusieurs mois à détecter l'attaque. Dans ce chapitre, nous proposons l'approche A^{2}SPICS pour *Applicative Attack Scenarios Production for Industrial Control Systems*. Cette approche est une analyse automatisée de propriétés de sûreté en présence d'attaquants. Comme décrit en section 2.3.3, elle se sert des résultats d'analyses de risques classiques en sécurité ne tenant pas compte des particularités des systèmes industriels, ainsi que d'analyses de risques en sûreté, fréquentes dans le monde OT. À partir des modèles d'attaquants issus de l'analyse de risques en sécurité, l'approche A^{2}SPICS vérifiera si ceux-ci sont en mesure de briser un sous-ensemble des propriétés de sûreté, jugées comme sensibles.

Sommaire

5.1. Contexte

Comme décrit en section 2.3.3, les employés contrôlant le système victime de l'attaque Maroochy Shire n'étaient visiblement pas préparés à réagir en cas d'attaque. On peut aisément s'en rendre compte du fait que non seulement l'attaque s'est déroulée sur plusieurs mois et que les conséquences étaient visibles dès les premières actions. Afin de préparer au mieux les opérateurs, il convient au préalable d'identifier les faiblesses du système. Pour cela, plusieurs méthodes d'analyse de risques ont été proposées au fil des années en sécurité et en sûreté de fonctionnement. Souvent manuelles, ces approches peuvent être automatisées grâce à des techniques de preuve et des tests. Preuve et test sont deux éléments indissociables des méthodes formelles. Ce chapitre traitant du test est donc le pendant du chapitre 4 qui s'est intéressé à la preuve de spécifications. Ce chapitre discute en particulier des analyses de risques en sécurité et en sûreté, de comment elles peuvent être utilisées conjointement et en partie automatisées afin de tester la présence d'attaques potentielles.

5.1.1. Test et preuve, deux méthodes complémentaires

Les définitions de preuves et tests varient selon les communautés mais on considère en général que la preuve a pour objectif de garantir une propriété tandis que le test vise à chercher des contre-exemples. En particulier, la notion de test peut s'appliquer à des critères différents (performances, régression, etc.). Nous nous focalisons sur des tests de robustesse (vérifier qu'une fonctionnalité respecte une propriété dans des conditions dégradées, dans notre cas lorsqu'un attaquant cherche à la saboter). Ainsi, les méthodes de preuve feront souvent des sur-approximations lors de l'analyse tandis que le méthodes de test feront des sous-approximations. Par exemple comme expliqué en section 4.2, dans le cadre d'analyses en présence d'attaquants, il est fréquent que le comportement de celui-ci soit non borné. Un outil faisant de la preuve aura donc tendance à sur-approximer le comportement d'un attaquant (le rendant donc plus puissant pour simplifier l'analyse), en lui donnant par exemple accès à plus de connaissances. De cette façon, si l'outil ne trouve pas d'attaques en présence de cet attaquant, alors la propriété restera vérifiée en présence d'attaquants plus faibles (non sur-approximé). Les attaques potentiellement trouvées pourront par contre résulter de l'approximation. A l'inverse, un outil faisant du test aura tendance à sous-approximer les capacités de l'attaquant. On sera alors certains que les attaques trouvées sont réelles mais l'outil pourra aussi en manquer. On s'aperçoit facilement que les deux approches ont de l'intérêt et gagnent à être utilisées conjointement.

5.1.2. Besoin d'analyses de risques

Les analyses de risques sont aujourd'hui utilisées, et intégrées dans le processus de développement, par une grande majorité des entreprises, industries ou encore dans le secteur public. Ces analyses de risques peuvent aussi bien traiter de la sûreté des procédés, de la santé des personnels ou, dans le cas de cette thèse, de la sécurité informatique. Cependant, contrairement à d'autres domaines, et dû à l'accélération du rythme des attaques informatiques ainsi qu'à l'évolution rapide des systèmes d'information, les analyses de risques en sécurité informatique doivent être rejouées régulièrement en ajustant certains paramètres. Par exemple, de nouvelles vulnérabilités doivent être considérées, de nouvelles normes doivent être adoptées, etc. Pouvoir rejouer ces analyses est par ailleurs très intéressant dans le cas des processus de certification ou d'audit afin d'accélérer ou faciliter le travail des auditeurs. Par ailleurs, différents points de vue peuvent être considérés dans les approches d'analyse de risques. Certaines méthodes pourront se focaliser sur les biens à protéger quand d'autres s'intéresseront d'abord aux sources de menaces.

5.1.3. Différents types d'attaquants

Par ailleurs, comme expliqué en section 1.3, de nombreuses vulnérabilités et exploits peuvent être décrits et trouvés sur Internet. Il existe également des entreprises qui achètent et revendent ces vulnérabilités au plus offrant. Ainsi, on doit maintenant faire face à une multitude d'attaquants avec des objectifs, des capacités techniques et des moyens très variables, allant des *script-kiddies* aux acteurs étatiques en passant par les hacktivistes, les mafias, ou encore les organisations terroristes. Il devient donc de plus

en plus intéressant de pouvoir considérer différents types d'attaquants dans les analyses de risques et les normes. Celles-ci intègrent donc la puissance des attaquants dans leurs analyses. Elles prennent par exemple en compte le niveaux d'expertise, le temps passé pour l'attaque, l'équipement requis ou encore la connaissance initiale. Cependant, se protéger d'attaquants puissants requiert souvent des coûts plus élevés et la sécurité informatique est parfois la variable d'ajustement des entreprises. Ainsi, suivant le contexte, se protéger d'attaquants plus faibles issus d'analyses de risques peut être un meilleur compromis par rapport à l'absence quasi totale de sécurité.

5.1.4. Les besoins particuliers des systèmes industriels

Dans le cas des systèmes industriels, nous avons déjà vu que les propriétés à garantir diffèrent des systèmes d'information. En particulier, ils doivent assurer des propriétés de sûreté et de disponibilité liées aux procédés qu'ils contrôlent. Par exemple le réacteur d'une centrale nucléaire ne doit pas dépasser une certaine température. De très nombreux travaux ont été développés pour identifier et vérifier ces propriétés. Cependant elles ne sont que très peu étudiées en présence d'attaquants. Cela vient notamment du fait que les systèmes industriels étaient historiquement séparés des réseaux non sûrs tels qu'Internet et n'étaient pas préparés à y être raccordés. La variété de protocoles de communication sécurisés ou non dans les réseaux industriels ainsi que la présence de plus en plus commune de composants sur l'étagère – *Commercials Off-The-Shelf* (COTS) – pouvant être facilement corrompus, encourage à analyser ces systèmes en considérant divers attaquants.

5.1.5. État de l'art

Les méthodes d'analyse de risques en sécurité informatique sont reconnues pour leur utilité dans le contexte des systèmes IT mais peinent encore à s'imposer dans le monde OT. Les analyses de risques en sûreté y sont par contre très importantes. Nous présentons dans un premier temps quelques exemples de méthodes d'analyse de risques en sûreté et en sécurité pour placer le contexte de ce chapitre qui propose de vérifier des propriétés de sûreté en présence d'attaquants. Dans un second temps, nous détaillons des travaux connexes.

5.1.5.1. Méthodes d'analyses de risques

Comme avancé en introduction, on peut distinguer les méthodes d'analyse de risques en deux catégories. D'un côté on trouve celles dédiées à la sûreté de fonctionnement, très utilisées dans les systèmes industriels tandis que de l'autre, on trouve celles dédiés à la sécurité informatique, très connues du monde IT. De nombreux travaux ont été entrepris pour lister, classifier et comparer ces méthodes d'analyse de risques avec par exemple l'étude de Pièvre-Cambacédès en 2010 [Pièvre-Cambacédès, 2010] ou encore les normes IEC/CEI 31010 [IEC-31010, 2009] et IEC/CEI 60300 [IEC-60300, 2003]. Une description de plusieurs méthodes connues et leur applicabilité aux systèmes industriels a également été faite par Knowles *et al.* en 2015 [Knowles et al., 2015] et Kabir-Querrec en 2017 [Kabir-Querrec, 2017]. On résume ici les méthodes les plus connues.

Méthodes dédiées à la sûreté : Ces méthodes plus anciennes ont pour la plupart été créées par des entreprises puis parfois normalisées. Elles sont particulièrement présentes dans le monde OT.

Analyse préliminaire de dangers : Mise au point dans les années 60 aux États-Unis par l'industrie des missiles, elle a ensuite été théorisée en 1988 par Villemeur [Villemeur, 1988]. Cette méthode vise à évaluer de façon macroscopique les dangers pouvant peser sur un système et leurs causes à l'aide de tableaux et de « check-list » conçus par des experts du domaine. Comme son nom l'indique, cette méthode constitue une analyse préliminaire à des études plus poussées.

AMDE/AMDEC : Aussi connue sous le nom de *Failure Mode and Effects Analysis* (FMEA) [IEC-60812, 1985], cette méthode a été développée aux États-Unis dans les années 60. De façon similaire à l'analyse préliminaire de dangers, l'idée est que les causes et les effets des défaillances possibles sont répertoriés au préalable dans des tableaux pour chaque domaine. En s'y référant on peut alors identifier et caractériser les conséquences des défaillances. Le « C » de AMDEC désigne la criticité afin de prendre en compte la gravité des conséquences. Très générique, cette méthode ne propose cependant pas de moyen pour croiser les cases des tableaux, ne considérant donc pas les failles multiples. Cette méthode a été standardisée en 1985 devenant FMEA.

HAZOP : HAZard and OPerability [IEC-61882, 2001] a été initialement proposé en 1977 par l'industrie de la chimie [CISHEC, 1977]. Cette méthode introduit l'utilisation de mots-clés (appelés *guide words*) tels que « Ne pas faire » ou « En plus de » afin de systématiser les analyses. Les résultats sont ensuite présentés sous forme de tableaux avec comme critères les déviations par rapport au comportement nominal, les causes, les conséquences, les actions à mener, la probabilité et la gravité. Cette méthode est standardisée en 2001.

IEC/CEI 61508 : La norme IEC/CEI 61508 [IEC-61508, 2010] a été proposée en 2010. C'est en partie cette norme qui est à l'origine de la norme IEC/CEI 62443 et ses *Security Levels*. La norme IEC/CEI 61508 propose, elle, les *Safety Integrity Levels* (SIL) afin de classifier les équipements et les logiciels en terme de criticité (quatre niveaux dans les deux cas). Différentes techniques et méthodologies, allant des spécifications jusqu'aux tests, sont prescrites selon le niveau à atteindre.

Méthodes dédiées à la sécurité informatique : Ces méthodes sont plus récentes et sont souvent issues d'agences gouvernementales (les industries étant historiquement peu enclines à adopter des mesures de sécurité informatique pour les raisons données ci-dessus). Elles sont beaucoup utilisées dans le monde IT et commencent à apparaître dans le monde OT. Ces méthodes d'analyses sont aujourd'hui portées par la norme ISO 27005 [ISO-27002, 2011] qui définit les grandes lignes d'une méthode d'analyse de risques en sécurité informatique.

EBIOS : Proposée en 1995 par l'ANSSI puis revue en 2010 [ANSSI, 2010], EBIOS pour Expression des Besoins et Identification des Objectifs de Sécurité est utilisée aussi bien par des acteurs publics que privés. Elle fonctionne en cinq phases étudiant respectivement le contexte, les événements redoutés, les menaces, les risques et enfin les mesures de sécurité. La méthode est fournie avec des exemples et un outil libre permettant de l'automatiser en partie [1].

MEHARI : D'abord proposé en 1998 par le CLUSIF (Club de la Sécurité des Systèmes d'Information Français) puis revue en 2010 [CLUSIF, 2010], MEHARI (pour Méthode Harmonisée d'Analyse des RIsques) est essentiellement utilisée dans le secteur privé. La méthode est fournie avec des exemples et un outil libre (sous forme de tableur) permettant de l'automatiser en partie [2].

Méthodes américaines : Plusieurs méthodes d'analyse de risques ont été développées aux États-Unis. La SP800-53 [Ross et al., 2013] a été proposée par le NIST et est essentiellement utilisée dans l'administration américaine. Elle a notamment inspiré son pendant sur les systèmes industriels, la norme SP800-82 [Stouffer et al., 2011]. MORDA [Buckshaw et al., 2005], pour *Mission Oriented Risk and Design Analysis* est une méthode développée par la NSA. OCTAVE [Alberts and Dorofee, 2002], pour *Operationally Critical Threat, Asset, and Vulnerability Evaluation* a été proposée en 1999 par le CERT-US [3] puis revue en 2005. Deux méthodes dérivées OCTAVE-S et OCTAVE Allegro sont proposées aux entreprises de plus petite taille. Enfin SQUARE, pour *Security QUAlity Requirements Engineering* a été développée en 2005 également par l'Université de Carnegie-Mellon [Mead and Stehney, 2005].

CRAMM Créé en 1985 par le CCTA (Central Communication and Telecommunication Agency) au Royaume-Uni, CRAMM (CCTA Risk Analysis and Management Method) a été revue en 2003. Cette méthode est utilisé par une très grande partie de l'administration anglaise ainsi que par l'OTAN.

Une liste très complète des méthodes d'analyse de risques en sécurité informatique est maintenue par l'ENISA [4]. En juillet 2017, la liste compte 17 méthodes et 30 outils pour aider à les appliquer. Il existe aussi une diversité de méthodes probabilistes telles que les arbres de défaillances (*Fault Trees*) [Ericson, 1999], les réseaux Bayésiens [Pearl, 1985], les modèles de Markov [Markov, 1906, Markov, 1971], ou encore les réseaux de Petri [Petri, 1962]. La plupart sont utilisées à la fois pour des analyses de sûreté et de sécurité ou ayant leur pendant suivant le contexte (par exemple les arbres d'attaques [Amoroso, 1994, Schneier, 1999] pour les arbres de défaillances). Les méthodes d'analyse de risques présentées dans cette section permettent la détection et potentiellement la quantification de la possibilité d'un risque (l'exploitation d'une vulnérabilité par une source de menace). La majorité des méthodes commencent par identifier les biens à protéger afin

1. https://www.ssi.gouv.fr/uploads/2011/10/Ebios_v2.zip
2. http://meharipedia.org/
3. Computer Emergency Response Team, Carnegie-Mellon University.
4. https://www.enisa.europa.eu/topics/threat-risk-management/risk-management/current-risk/risk-management-inventory

de définir les limites du périmètre de l'analyse. Il est néanmoins possible que ces limites soient définies en fonction d'autres paramètres, en débutant par exemple l'analyse par les sources de menaces (en particulier les attaquants). Par ailleurs, on retrouve des concepts similaires entre les analyses de risques en sûreté de fonctionnement et en sécurité (par exemple la dualité entre les attaques et les défaillances). Les méthodes d'analyse de risques présentées dans cette section peuvent être utilisées comme sources d'information pour des analyses plus spécifiques comme présentées en section 5.1.5.2.

5.1.5.2. Approches évaluant la sécurité de systèmes industriels

L'évaluation des risques en matière de sécurité informatique pour les systèmes industriels est un domaine récent mais en plein essor. Depuis Byres *et al.* en 2004 [Byres et al., 2004], de nombreuses méthodes ont été proposées avec des spécificités très variées. En 2015, Cherdantseva *et al.* [Cherdantseva et al., 2015] ont proposé un état de l'art sur 24 méthodes d'analyse de risques en sécurité informatique pour les systèmes industriels, publiées entre 2004 et 2014. Les auteurs classifient ces méthodes avec des critères tels que le domaine d'application, l'utilisation ou non de probabilités dans les évaluations (si oui les sources des distributions), l'application à des études de cas ou encore l'outillage de la méthode. Des états de l'art similaires ont été publiées en 2012 par Piètre-Cambacédès et Bouissou [Piètre-Cambacédès and Bouissou, 2013], et en 2015 par Kriaa *et al.* [Kriaa et al., 2015b]. On résume ci-après un panel des travaux listés dans ces états de l'art, choisis soit pour leur notoriété dans le domaine soit pour montrer la variété des approches possibles.

Byres *et al.* En 2004, Byres *et al.* [Byres et al., 2004] proposent une approche qualitative reposant sur des arbres d'attaques pour évaluer la sécurité des systèmes industriels. Leur approche est focalisée sur des systèmes utilisant le protocole MODBUS et visant en particulier le domaine électrique. Ils prennent en compte l'architecture du système, les contre-mesures mises en place, la difficulté de l'attaque, la probabilité de détection et calculent son impact en terme de coût, les buts des attaquants étant les feuilles de l'arbre. Les paramètres sont mesurés sur une échelle de un à quatre. Ils étendent cette approche en 2006 et l'outillent dans [Byres et al., 2006].

McQueen *et al.* En 2006, McQueen *et al.* [McQueen et al., 2006] introduisent une approche basée sur une formule pour mesurer les risques sur les SCADA de faibles tailles. Cette formule prend en compte la probabilité que le système soit visé par une attaque, la probabilité que l'attaque se fasse, la probabilité que le système présente une faille, la probabilité que l'attaque réussisse et la probabilité que l'attaque endommage le système. Ils utilisent pour cela des graphes de compromissions où les distributions de probabilités sont supposées être données par des experts et instancient leur méthode sur un système industriel. Cette approche n'est pas outillée et ne tient pas compte de la logique applicative du système testé.

Patel *et al.* En 2008, Patel *et al.* [Patel et al., 2008] détaillent une approche similaire prenant en compte à la fois l'impact financier de l'attaque et la vulnérabilité du système. Ces deux indices sont représentés par des valeurs comprises entre un et cent

et sont propagées au sein d'un arbre de vulnérabilité. Les distributions de probabilité sont également supposées données par des experts. Ils montrent le fonctionnement de leur méthode sur un exemple avec une cuve et une pompe (similaire à celui présenté en annexe A.4). Cette approche n'est pas outillée.

Ten *et al.* En 2010, Ten *et al.* [Ten et al., 2010] proposent une méthode quantitative basée sur des graphes d'attaques. Ils se basent sur la probabilité qu'un équipement soit compromis ainsi que sur la probabilité des scénarios d'attaques. Ils mesurent les probabilités au moyen d'historiques de données (par exemple sur de précédentes attaques) ainsi que sur des tests d'intrusions et les politiques de sécurité de l'entreprise. Leur approche est centrée sur le domaine électrique et un outil est mentionné sans plus d'informations.

Cárdenas *et al.* En 2011, Cárdenas *et al.* [Cárdenas et al., 2011] introduisent une formule permettant de calculer le risque et l'impact de la compromission de capteurs dans des réseaux. Ils calculent ensuite la divergence des valeurs reçues par rapport aux valeurs attendues au moyen d'un modèle linéaire. Aucune mention n'est faite sur la source des distributions de probabilités. Ils proposent un outil pour leur approche, formé d'un modèle MATLAB, et l'applique sur un cas d'étude issu d'un réacteur chimique.

Kriaa *et al.* En 2012, Kriaa *et al.* [Kriaa et al., 2012] présentent une méthode basée sur des arbres de fautes combinés avec des processus de Markov. Ils permettent notamment l'utilisation de portes logiques (ET et OU) entre les noeuds de l'arbre. Ils utilisent l'outil KB3 [Piètre-Cambacédès et al., 2011] et appliquent leur approche sur l'attaque Stuxnet. En 2015, ils publient S-CUBE [Kriaa et al., 2015a], une méthode qui reprend l'approche originale dans le langage Figaro. Cette approche prend en compte le comportement nominal du système à vérifier. Les probabilités utilisées dans les deux approches sont « basées sur leurs propres estimations et des documents de consultants en sécurité » [5] [Kriaa et al., 2012].

Rocchetto et Tippenhauer En 2017, Rocchetto et Tippenhauer [Rocchetto and Tippenhauer, 2017] présentent une méthode basée sur l'outil de vérification de protocoles cryptographiques CL-Atse [Turuani, 2006]. Ils modélisent pour cela le système testé en tenant compte de la logique applicative du procédé à l'aide du langage ASLAN++ [Armando et al., 2012] créé dans le cadre du projet AVANTSSAR [6]. Les auteurs considèrent un attaquant proche de l'intrus Dolev-Yao, notamment étendu pour prendre en compte les interactions physiques possibles avec le procédé [Rocchetto and Tippenhauer, 2016]. Des propriétés LTL sur l'état global du système sont spécifiées et traduites par ASLAN++ en des propriétés pouvant être vérifiées par CL-Atse. Ils appliquent leur méthode sur un système contrôlant une cuve (très similaire au système victime de l'attaque Maroochy Shire).

11 des 24 méthodes listées dans Cherdantseva *et al.* s'appliquent explicitement au domaine électrique. Les auteurs remarquent également que 17 méthodes sur les 24 ne

5. Traduction de « [...] based on our own estimation and writings by security consultants. »
6. http://www.avantssar.eu

sont pas outillées, quatre proposent un nouvel outil dédié à la méthode et trois se basent sur des outils existants, tels que MATLAB ou FORTRAN. De plus, les méthodes basées sur des distributions de probabilités ne donnent que très peu d'informations sur les sources de ces distributions et sur leurs pertinences. Dans tous les cas, très peu d'informations sont données sur les conditions d'expérimentations. Comme explicité par Kabir-Querrec [Kabir-Querrec, 2017], cela montre le manque de maturité du domaine et l'importance d'améliorer les méthodes pour évaluer les risques de sécurité informatique sur les systèmes industriels. On peut remarquer par ailleurs que les approches prennent en compte soit la topologie du système en s'intéressant à la position des attaquants et à leurs capacités, soit à la logique applicative du procédé mais ne considèrent que très peu ces deux aspects ensemble.

5.2. Description de l'approche A²SPICS

Pour répondre à la problématique des analyses de sécurité dans le contexte des systèmes industriels, en tenant compte à la fois des attaquants et de la logique applicative du procédé, nous proposons l'approche A²SPICS pour *Applicative Attack Scenarios Production for Industrial Control Systems*. Cette approche est une analyse automatisée visant à vérifier des propriétés de sûreté en présence d'attaquants. Comme décrit en section 2.3.3, notre approche fonctionne en deux phases. Dans une première phase une analyse de risques en sécurité décrit les attaquants possibles. En parallèle, une analyse de risques en sûreté de fonctionnement (considérée comme hors de l'approche car usuelle dans le monde OT) définit notamment les propriétés de sûreté de fonctionnement que le système doit respecter. À partir des attaquants issus de la phase 1, la phase 2 vérifie si ceux-ci sont en mesure de mettre en défaut certaines des propriétés de sûreté, jugées comme particulièrement critiques. En particulier, nous considérons des propriétés portant sur les valeurs des variables du système (de façon similaire aux règles de filtrage présentés en chapitre 3). Par exemple dans le cas de l'attaque Maroochy Shire (annexe A.3), la pompe ne devrait pas recevoir une commande d'arrêt si le capteur détecte du liquide.

Contributions : Les contributions de l'approche A²SPICS sont les suivantes :

- Phase 1 : Une méthode d'analyse de risques en sécurité, focalisée sur le point de vue des attaquants (publication à la conférence AFADL 2016 [Puys et al., 2016b]).
- Phase 2 : La production de scénarios d'attaques contre des propriétés de sûreté de fonctionnement en présence d'attaquants (publication à la conférence FPS 2017 [Puys et al., 2017]).
- L'application de l'approche sur une étude de cas, implémentée avec le model-checker UPPAAL. D'autres exemples (non formalisés) sont proposés en annexe A.
- Une étude approfondie sur l'intérêt et la possibilité d'utiliser des outils de vérification de protocoles cryptographiques dans le cadre de l'approche A²SPICS.

Plan du chapitre : Le reste de cette section présente la terminologie utilisée ainsi qu'une étude de cas sur laquelle sera appliquée l'approche A²SPICS, puis décrit l'ap-

proche elle-même. Ensuite, la section 5.3 décrit la phase 1 de l'approche : une méthode d'analyse de risques en sécurité focalisée sur les attaquants. La section 5.4 décrit la phase 2 de l'approche : une méthode de vérification de propriétés de sûreté en présence d'attaquants et son implémentation avec le model-checker UPPAAL. La section 5.5 discute de la possibilité d'implémenter l'approche via des outils de vérification de protocoles cryptographiques comme ProVerif et Tamarin. Enfin, la section 5.6 conclut et dresse les perspectives de l'approche.

5.2.1. Terminologie utilisée

Les modèles analysés par notre approche sont décrits par plusieurs paramètres. Nous définissons ici la terminologie qui sera utilisée dans le reste du chapitre.

Procédé : C'est l'application industrielle qui est contrôlée par le système. Le procédé peut par exemple réaliser de la production ou de la distribution d'électricité, du traitement d'eaux usées ou encore contrôler des moyens de transports. Comme décrit en section 2.1, le procédé est composé de *variables* présentes sur les serveurs (par exemple les automates programmables). Les évolutions possibles du procédé, impactant les valeurs des variables, sont appelées le *comportement* du procédé.

Clients : Les clients sont les entités envoyant des commandes pour monitorer et contrôler le procédé. Suivant le niveau de la classification de Purdue dans lequel les clients se trouvent, ils peuvent représenter des IHM, des clients SCADA, etc. (les serveurs d'un niveau pouvant être vus comme les clients du niveau suivant).

Serveurs : Les serveurs sont les entités recevant des commandes des clients et modifiant le procédé en conséquence. De même que les clients, ils peuvent représenter aussi bien des serveurs SCADA que des automates programmables suivant le niveau de la classification de Purdue dans lequel ils se trouvent. Les serveurs implémentent un protocole que les clients utilisent pour envoyer leurs commandes (par exemple MODBUS ou OPC-UA). Ces protocoles pouvant offrir des fonctionnalités de sécurité différentes, leur prise en compte sera cruciale dans le reste de l'approche.

Composants : Les composants sont toutes les entités envoyant et recevant des messages, cela signifie les clients, les serveurs et les attaquants.

Canaux de communication : Un canal de communication est un lien réseau entre deux composants, utilisé pour transmettre des messages. Les messages sont formatés dans un protocole de transport tel que défini en section 2.2.1. Les fonctionnalités de sécurité proposées par le protocole (chiffrement, signature du message, redondance, etc.) définissent la sécurité du canal.

Topologie : L'ensemble des composants communiquant par des canaux de communication est appelé la topologie du système. Dans le cadre de cette thèse, nous considérons les topologies dans un réseau. Cependant, l'approche peut être portée au sein de tout

système divisé en plusieurs entités (par exemple le dispositif de filtrage présenté en chapitre 3 avec son cœur et ses guichets, ou encore un processeur multicœur) et sur les communications entre ces entités.

Propriétés de sûreté : Les propriétés de sûreté que le système doit garantir sont le résultat d'analyses de risques en sûreté. Parmi ces propriétés, certaines sont considérées comme vitales pour le procédé et/ou pour l'intégrité des humains et de l'environnement (par exemple ne pas ouvrir la porte du four si celui-ci est en fonctionnement). Ce sont donc ces propriétés que les attaquants tenteront de mettre en défaut.

Attaquants : Les attaquants sont les intrus actifs ou passifs essayant de violer les propriétés. Ils sont paramétrés d'une part par leur position dans la topologie du système et d'autre part par leurs capacités et comportements. Leur position détermine avec quels clients et serveurs ils peuvent communiquer (donc sur quels canaux ils se trouvent). Leurs capacités et leurs comportements indiquent quels types d'actions ils peuvent accomplir (par exemple modifier un message ou le rejouer) et sous quelles conditions. Suivant leurs capacités, les attaquants peuvent mémoriser les messages vus. Dans ce cas, cette connaissance croit de façon monotone et peut être utilisée pour *déduire* de nouvelles connaissances (de façon analogue à l'intrus Dolev-Yao présenté en section 4.2.1). Les notions de capacités et connaissances initiales de l'attaquant sont des paramètres classiques des guides, normes et analyses de risques en sécurité au même titre que son équipement ou le temps qu'il peut passer à attaquer le système.

5.2.2. Une étude de cas

Pour illustrer l'intérêt de l'approche A²SPICS, nous proposons de l'appliquer sur un exemple qui provient du simulateur de procédé VirtualPlant[7]. Des bouteilles avancent sur un tapis roulant. Un capteur détecte lorsqu'elles sont en position, arrête le tapis roulant et actionne une vanne faisant couler le liquide. Un autre capteur permet de détecter lorsque la bouteille sous la vanne est pleine. La vanne se referme, et le tapis roulant redémarre.

Plusieurs propriétés peuvent être envisagées, telles que le fait que la vanne ne s'ouvre que lorsqu'une bouteille est en position et que le tapis roulant soit à l'arrêt. Une propriété complémentaire pourrait être que les bouteilles ayant dépassé la vanne soient bien remplies. On définit donc les propriétés suivantes :

Propriété 1 : La vanne ne s'ouvre que lorsqu'une bouteille est en position.
Propriété 2 : Le moteur du tapis roulant ne démarre que lorsque la bouteille en position est pleine.
Propriété 3 : La vanne ne s'ouvre que lorsque le moteur est arrêté.

Malgré sa simplicité, cet exemple permet un large éventail d'instanciations possibles. Premièrement, les différentes propriétés à garantir peuvent être variées (trois propriétés critiques sont données ci-dessus mais on pourrait aussi considérer des propriétés annexes comme limiter l'usure du matériel). Par ailleurs, différentes topologies du système

7. https://github.com/jseidl/virtuaplant

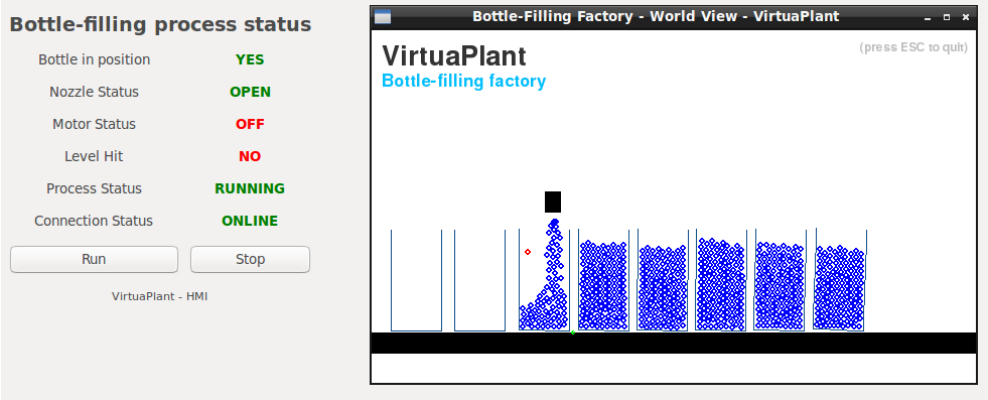


FIGURE 5.1. – Chaîne de mise en bouteille

peuvent être envisagées. Nous considérons dans cet exemple le tapis roulant et la vanne comme deux composants distincts. Ils pourraient être contrôlés par le même serveur ou bien chacun par un serveur dédié comme représenté dans la figure 5.2. Enfin, les protocoles de communication utilisés peuvent avoir différents niveaux de sécurité, étant donc plus ou moins vulnérables à des attaquants qui eux peuvent être placés à diverses positions de la topologie. Par exemple un attaquant pourrait être placé sur un canal de communication (comme dans la figure 5.2) ou bien comme un équipement corrompu (par exemple un client ou serveur légitime, contrôlé par un opérateur malveillant ou infecté par un virus).

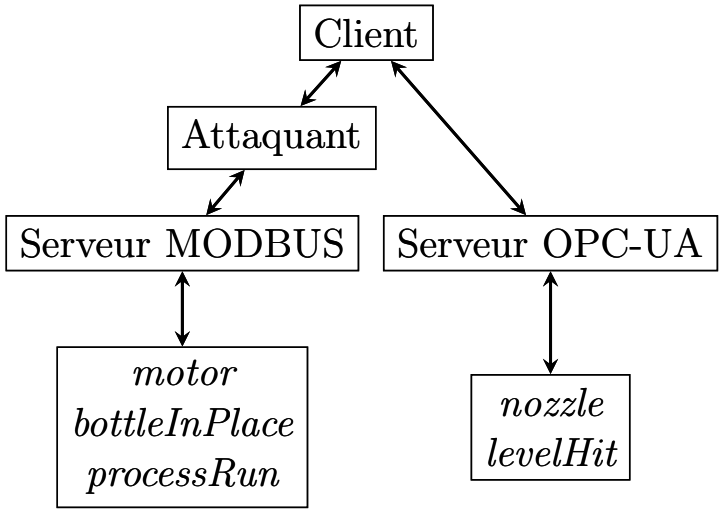


FIGURE 5.2. – Exemple de topologie réseau

5.2.3. Fonctionnement de l'approche

La figure 5.3 illustre le principe de l'approche A²SPICS. Cette approche fonctionne en deux phases : la phase 1 étant une analyse de risques focalisée sur les attaquants et

la phase 2 une méthode de génération de scénarios d'attaques applicatives.

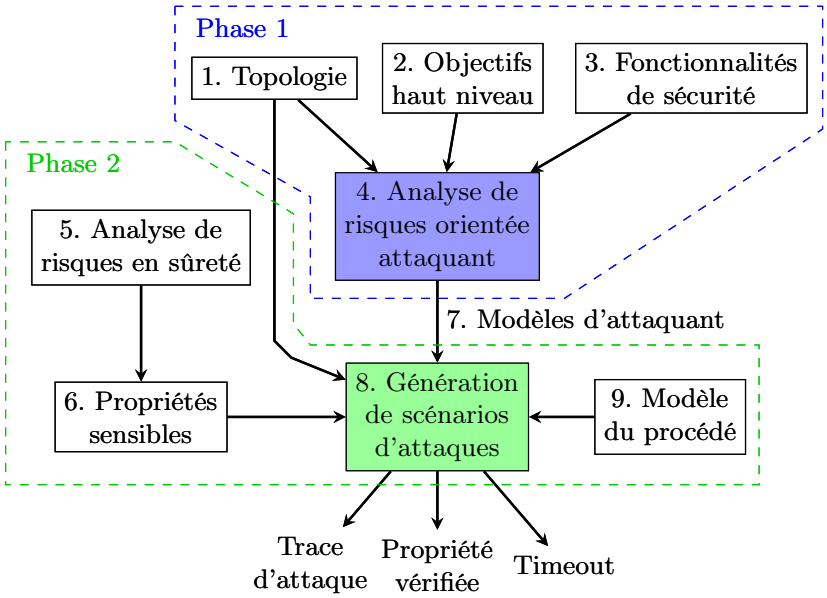


FIGURE 5.3. – L'approche A²SPICS

Phase 1 : Dans une première phase, on propose une analyse de risques en sécurité orientée attaquant (4). Cette analyse est basée sur la topologie (1), les objectifs des attaquants potentiels (2), ainsi que les fonctionnalités de sécurité apportées par les protocoles de communication (3). Les objectifs considérés ici sont des notions de haut niveau (telles que « altérer un message » ou « contourner l'authentification »). Ces objectifs représentent des attaques en sécurité consistant à exploiter une vulnérabilité. Ils sont aussi souvent référencés sous le terme de « menace » dans les cibles de sécurité et les profils de protection. A l'issue de cette analyse, des scénarios d'attaques sont produits, décrivant comment un attaquant peut atteindre un de ses objectifs (par exemple en tirant parti de potentielles faiblesses des protocoles de sécurité). Cette première partie de l'approche est décrite plus en détail en section 5.3.

Phase 2 : Dans une seconde phase, on s'intéresse à vérifier si des attaquants peuvent mettre en défaut des propriétés de sûreté de fonctionnement (le but principal de l'approche). Cette analyse prend en entrée : des propriétés de sûreté jugées comme critiques (6) à l'issue d'une analyse de risques en sûreté (5), des *modèles d'attaquants* (7) issus de la phase 1, et un modèle du procédé (9). Les propriétés de sûreté que le système doit garantir seront considérées comme des buts pour les attaquants. Enfin, le procédé est modélisé afin de simuler les conséquences possibles des actions de l'attaquant. Par exemple dans le cas d'un haut fourneau, une propriété de sûreté est que le four ne démarre pas si la porte est ouverte. Via une méthode de génération de scénarios d'attaques (8), on peut alors savoir, pour chaque propriété, si un attaquant peut la violer et, le cas échéant, produire une trace d'exécution menant à l'attaque. La génération peut

aussi être inconclusive pour des raisons propres à l'outil utilisé (par exemple en cas de timeout).

Interactions entre les deux phases : Les modèles d'attaquants utilisés en phase 2 sont constitués de : (i) la position de l'attaquant dans la topologie, (ii) ses capacités, et (iii) ses connaissance initiales. Ces caractéristiques sont tirées des attaquants identifiés dans la phase 1. Comme expliqué précédemment, la position de l'attaquant peut être soit sur un canal de communication (en MITM), soit comme un équipement corrompu (un composant contrôlé par un opérateur malveillant ou infecté par un virus). Les capacités de l'attaquant décrivent d'une part les actions qu'il pourra effectuer pour influer sur le réseau (par exemple forger un message), et d'autre part son système de déduction (utiliser ses connaissances pour en déduire de nouvelles, par exemple déchiffrer un message). Dans le cadre de la phase 1, les capacités d'un attaquant sont directement liées aux fonctionnalités de sécurité proposées par les protocoles de communication (par exemple le chiffrement). Les connaissances initiales de l'attaquant sont tirées de sa position (un attaquant sur composant corrompu connaîtra les clés cryptographiques du composant, un serveur connaîtra les variables qu'il peut manipuler).

5.3. Phase 1 : Analyse de risques orientée attaquant

Dans cette section, nous proposons une méthode d'analyse de risques focalisée sur le point de vue de l'attaquant. En effet, les méthodes connues telles que celles citées en section 5.1.5.1 sont le plus souvent focalisées sur les biens à protéger. Cependant, dans le cadre de l'approche A²SPICS, il nous parait intéressant que cette analyse de risques en sécurité permette de déterminer quelles sont les positions possibles des attaquants dans la topologie et quelles sont leurs capacités. La méthode proposée en phase 1, résumée en figure 5.4, se décompose en deux étapes. En section 5.3.1, nous étudions dans un premier temps les objectifs haut niveau des attaquants et comment ils pourraient les réaliser (partie descendante). Dans un second temps, en section 5.3.2 nous déterminons leurs capacités de nuisance en fonction des protections apportées par les protocoles de communication (partie ascendante). Les deux partie de la méthode d'analyse convergent pour donner des scénarios d'attaques dans lesquels un attaquant parvient à réaliser ses objectifs.

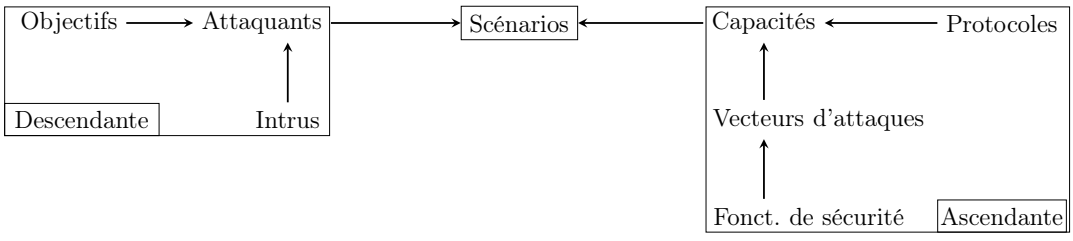


FIGURE 5.4. – Schéma de la méthode d'analyse de risques

5.3.1. Approche descendante

Nous commençons par définir l'ensemble des attaquants $\mathcal{A}$ et l'ensemble des objectifs d'attaques $\mathcal{O}$. Les objectifs considérés sont des notions de haut niveau (telles que « altérer un message » ou « contourner l'authentification »). Ces ensembles sont liés par la fonction $Obj(a)$, définissant l'ensemble des objectifs de $\mathcal{O}$ retenus pour un attaquant a dans l'analyse de risques. Cette fonction doit tenir compte de la position des attaquants dans la topologie du système (par exemple un attaquant ne devrait pas avoir pour objectif la modification d'un message s'il n'y a jamais accès).

Exemple :

Nous considérons la topologie en figure 5.5 où le $Client_1$ communique avec le $Serveur_1$ via le protocole OPC-UA configuré en mode None (plus de détails en section 2.2.3.2). Similairement, le $Client_A$ communique avec le $Serveur_2$ en FTP_{Auth} (le protocole FTP avec une authentification par mot de passe). Un routeur est positionné au milieu des clients et des serveurs. Les attaquants (en couleur) sont $\mathcal{A} = \{Client_A, Routeur_A\}$ (ici un client et un routeur compromis) et les objectifs d'attaques[8] considérés, sont $\mathcal{O} = \{VolDonnees, ContAuth, Alte, Alte_C\}$ avec :

- $VolDonnees$ = Vol de données ;
- $ContAuth$ = Contournement d'authentification ;
- $Alte$ = Altération d'un message ;
- $Alte_C$ = Altération ciblée d'un message.

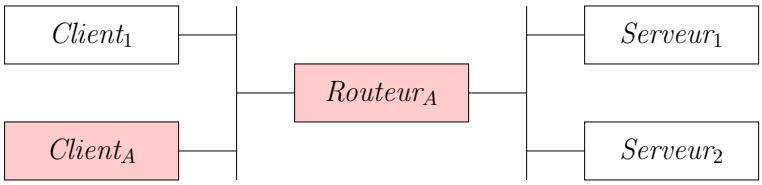


FIGURE 5.5. – Exemple de topologie (attaquants en rouge)

Les attaquants $\mathcal{A}$ et les objectifs $\mathcal{O}$ sont liés par la fonction $Obj(a)$ définie en table 5.1, où un ✓ signifie que l'objectif o est retenu pour l'attaquant a.

$Obj(a)$	$VolDonnees$	$ContAuth$	$Alte$	$Alte_C$
$Client_A$	✗	✓	✗	✗
$Routeur_A$	✓	✓	✓	✓

TABLE 5.1. – Exemple d'objectifs retenus pour chaque attaquant

Nous définissons ensuite l'ensemble des vecteurs d'attaques $\mathcal{V}$, et $Real(o) \subseteq \mathbb{P}(\mathcal{V})$, la fonction entre un objectif et l'ensemble des parties de $\mathcal{V}$ (i.e. : les combinaisons de vecteurs d'attaques). Ainsi, $Real(o)$ décrit la réalisation d'un objectif o à l'aide de

8. Ici tirés des profils de protection de l'ANSSI.

vecteurs de $\mathcal{V}$. Un vecteur peut être vu comme les techniques, tactiques ou procédés pouvant servir à la réalisation d'une attaque [Conchon and Caire, 2015].

Exemple :

Nous considérons ici les vecteurs $\mathcal{V} = \{Lire, Usurp, Mod, Rej\}$ avec :

- $Lire$ = Lecture (et compréhension) d'un message
- $Usurp$ = Usurpation d'une identité
- Mod = Modification d'un message
- Rej = Rejeu d'un message

La fonction *Real* décrivant comment réaliser les objectifs d'attaque à l'aide des vecteurs d'attaques peut par exemple être :

- $Real(VolDonnees) = \{\{Lire\}\}$
- $Real(ContAuth) = \{\{Usurp\}, \{Rej\}\}$
- $Real(Alte) = \{\{Mod\}\}$
- $Real(Alte_C) = \{\{Lire, Mod\}\}$

En particulier, *ContAuth* peut être réalisé en usurpant une identité **ou** en rejouant un message d'authentification (par exemple l'envoi d'un mot de passe). À l'inverse, $Alte_C$ nécessite **à la fois** la capacité de modifier un message et d'en comprendre le contenu.

Ainsi, nous sommes en mesure de représenter les attaquants, leurs objectifs d'attaques et comment ces objectifs peuvent être réalisés au moyen de vecteurs d'attaques. La modélisation des objectifs et de leurs réalisations est générique dans l'approche proposée. L'ensemble des objectifs possibles $\mathcal{O}$ et la fonction *Real* peuvent être prédéfinis dans un guide de bonnes pratiques ou pour un audit, mais peuvent aussi être spécifiés de façon personnalisée pour chaque analyse. Par exemple, dans le cas d'un protocole de communication, le contournement de l'authentification peut se faire en rejouant un message (comme dans le cas de FTP_{Auth}). Or, dans le contexte d'une analyse d'un composant contre des attaques physiques, l'authentification peut être contournée au moyen d'une variation de la tension électrique du composant : $Real(ContAuth) = \{\{VariationTension\}\}$. La section suivante décrit comment une analyse des fonctionnalités de sécurité assurées par les protocoles vérifie si ces objectifs d'attaques sont faisables.

5.3.2. Approche ascendante

Dans un second temps, nous définissons l'ensemble des configurations des protocoles $\mathcal{P}$ considérés pour l'analyse. Dans la suite, nous considérerons chaque configuration comme un protocole différent. On définit $Vect(p) \subseteq \mathcal{V}$ comme l'ensemble des vecteurs d'attaques de $\mathcal{V}$ accessibles à l'attaquant pour un protocole $p \in \mathcal{P}$ (i.e. : les faiblesses du protocole dues à l'absence de fonctionnalités de sécurité). Ces faiblesses peuvent notamment être déterminées grâce aux techniques de vérification de protocoles cryptographiques utilisées au chapitre 4.

Exemple :

Soient les protocoles $\mathcal{P} = \{\text{MODBUS}, \text{FTP}, \text{FTP}_{Auth}, \text{FTPS}, \text{OPC-UA}_{None}, \text{OPC-UA}_{Sign}, \text{OPC-UA}_{SignEncrypt}\}$. Les vecteurs d'attaques $Vect(p)$ pour chaque protocole $p \in \mathcal{P}$ sont définis en table 5.2, où un ✓ signifie que le protocole présente une faiblesse rendant le vecteur d'attaque est accessible à l'attaquant. Les protocoles MODBUS, FTP et OPC-UA_{None} ne garantissent aucune sécurité et permettent donc tous les vecteurs d'attaques. Le protocole FTP_{Auth} ajoute une authentification à l'aide d'un mot de passe empêchant l'usurpation d'identité tandis que FTPS est le protocole FTP sécurisé à l'aide de TLS ou SSL. Les protocoles OPC-UA_{Sign} et $\text{OPC-UA}_{SignEncrypt}$ apportent des signatures cryptographiques et de l'estampillage aux messages, rendant ainsi impossible leur usurpation, modification ou rejeu. Enfin $\text{OPC-UA}_{SignEncrypt}$ garantit également la confidentialité des communications (plus de détails sur la sécurité de MODBUS et OPC-UA en section 2.2.3 et en chapitre 4).

$Vect(p)$	*Lire*	*Usurp*	*Mod*	*Rej*
MODBUS	✓	✓	✓	✓
FTP	✓	✓	✓	✓
FTP_{Auth}	✓	✗	✓	✓
FTPS	✗	✗	✗	✗
OPC-UA_{None}	✓	✓	✓	✓
OPC-UA_{Sign}	✓	✗	✗	✗
$\text{OPC-UA}_{SignEncrypt}$	✗	✗	✗	✗

TABLE 5.2. – Exemple de vecteurs d'attaques pour chaque protocole

Il est alors possible de déterminer si un objectif o est réalisable à l'aide de l'ensemble des vecteurs d'attaques pour un protocole p en vérifiant si : $\exists v \in Real(o) \mid v \subseteq Vect(p)$. C'est-à-dire, s'il existe un ensemble de vecteurs d'attaques réalisant un objectif (par exemple $\{Lire, Mod\} \in Real(Alte_C)$), dont tous les vecteurs seraient accessibles pour un protocole donné (par exemple $\{Lire, Mod\} \subseteq Vect(\text{OPC-UA}_{None})$).

L'ensemble des scénarios d'attaques $\mathcal{S}_{a,p}$ pour un attaquant a et un protocole p est alors défini par :

$$\mathcal{S}_{a,p} = \{(o, v) \mid o \in Obj(a) \wedge v \in Real(o) \wedge v \subseteq Vect(p)\}$$

L'ensemble des scénarios d'attaques pour un attaquant et un protocole est donc toutes les réalisations des objectifs de l'attaquant qui sont permises par le protocole. Les scénarios d'attaques à considérer dans le cadre d'une campagne de test sont l'ensemble des $\mathcal{S}_{a,p}, \forall a \in \mathcal{A}$ et $\forall p \in \mathcal{P}$ que pourrait utiliser chaque attaquant.

Exemple :

Dans notre exemple, les objectifs réalisables pour chaque protocole sont donnés dans la table 5.3. On remarque que contrairement à la table 5.2, la case *ContAuth* de la ligne FTP_{Auth} est cochée. Cela vient du fait que même si $Usurp \notin Vect(\text{FTP}_{Auth})$, on a

$Real(o)$	$VolDonnees$	$ContAuth$	$Alte$	$Alte_C$
MODBUS	✓	✓	✓	✓
FTP	✓	✓	✓	✓
FTP_{Auth}	✓	✓	✓	✓
FTPS	✗	✗	✗	✗
OPC-UA_{None}	✓	✓	✓	✓
OPC-UA_{Sign}	✓	✗	✗	✗
$\text{OPC-UA}_{SignEncrypt}$	✗	✗	✗	✗

TABLE 5.3. – Exemple de réalisation des objectifs pour chaque protocole

$\{Rej\} \in Real(ContAuth)$ et $Rej \in Vect(\text{FTP}_{Auth})$. Ainsi le client $Client_A$, communiquant via le protocole FTP_{Auth}, peut réaliser l'objectif $ContAuth$ via le vecteur d'attaque Rej. L'ensemble des scénarios à considérer pour cet attaquant est donc :

$$\mathcal{S}_{Client_A,\text{FTP}_{Auth}} = \{(ContAuth, \{Rej\})\}$$

De même pour $Routeur_A$ qui communique à la fois avec les protocoles OPC-UA_{None} et FTP_{Auth} :

$$\mathcal{S}_{Routeur_A,\text{OPC-UA}_{None}} = \{(VolDonnees, \{Lire\}), (ContAuth, \{Usurp\}), (ContAuth, \{Rej\}), (Alte, \{Mod\}), (Alte_C, \{Lire, Mod\}))\}$$

$$\mathcal{S}_{Routeur_A,\text{FTP}_{Auth}} = \{(VolDonnees, \{Lire\}), (ContAuth, \{Rej\}), (Alte, \{Mod\}), (Alte_C, \{Lire, Mod\})\}$$

L'ensemble des scénarios d'attaques issus de cette analyse et qui doivent être évités est donc : $\mathcal{S}_{Client_A,\text{FTP}_{Auth}}$, $\mathcal{S}_{Routeur_A,\text{OPC-UA}_{None}}$ et $\mathcal{S}_{Routeur_A,\text{FTP}_{Auth}}$.

L'analyse de risques proposée dans cette section est basée sur la topologie, les objectifs des attaquants potentiels, ainsi que les fonctionnalités de sécurité apportées par les protocoles de communication. A l'issue de cette analyse, des scénarios d'attaques sont produits, décrivant comment un attaquant peut atteindre ses objectifs en tirant parti de potentielles faiblesses de sécurité des protocoles de communication. Comme expliqué ci-dessus, toutes les notions modélisées sont génériques. Les tables de réalisations des objectifs, des faiblesses des protocoles, etc peuvent être prédéfinies et réutilisables mais aussi être personnalisés pour chaque analyse. Nous allons voir en section 5.4 comment tirer parti des attaquants identifiés en phase 1 (en termes de positions et capacités) pour vérifier des propriétés de sûreté en phase 2.

5.4. Phase 2 : Génération de scénarios d'attaques

Dans le cadre du mémoire de master d'Abdelaziz Khaled [Khaled, 2017], nous nous sommes intéressés à étendre la méthode d'analyse de risques décrite en section 5.3 à

des propriétés de sûreté de fonctionnement. Nous appelons cette analyse la phase 2 de l'approche A^{2}SPICS. Cette phase prend en entrée d'une part des attaquants identifiés dans la phase 1, et d'autre part un modèle du comportement du système avec des propriétés de sûreté qu'il doit respecter, issues d'une analyse de risques en sûreté. Nous vérifions alors si ces propriétés (garanties par le système en temps normal) restent sûres en présence d'attaquants. Les attaquants sont modélisés par leurs positions dans la topologie, leurs capacités, et leurs connaissances initiales. Dans cette phase de l'analyse, les propriétés considérés sont des propriétés fonctionnelles jugées critiques pour le système. Par exemple dans le cas d'un haut fourneau, une propriété de sûreté est que le four ne démarre pas si la porte est ouverte. L'analyse réalisée dans la phase 2 a été automatisée via le model-checker UPPAAL [9]. La description de l'approche A^{2}SPICS et l'implémentation avec UPPAAL ont fait l'objet d'une publication à la conférence FPS 2015 [Puys et al., 2017]. Nous décrivons d'abord en section 5.4.1 le fonctionnement de la phase 2 et son implémentation avec UPPAAL. Ensuite, nous détaillons les résultats trouvés par l'outil sur l'étude de cas en section 5.4.2.

5.4.1. Description des éléments modélisés

La modélisation se divise en deux parties principales : (i) le modèle du système, et (ii) les composants. Enfin, la figure 5.6 schématise ces paramètres des modèles.

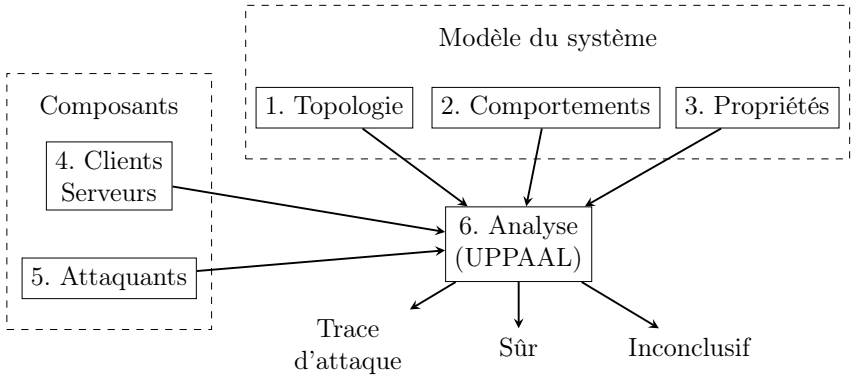


FIGURE 5.6. – Modélisation avec UPPAAL

Le modèle du système est composé de la topologie (1), des comportements des composants (clients et serveurs, en particulier les interactions entre les serveurs et le procédé), et les propriétés de sûreté à garantir (3). A cela vient s'ajouter la modélisation des clients et serveurs (4) et des attaquants (5). Les clients et serveurs implémentent un protocole de communication. Ils sont conçus pour être réutilisables d'une analyse à l'autre et des clients et serveurs MODBUS et OPC-UA sont ainsi fournis à l'utilisateur. De façon similaire, les attaquants sont issus de la phase 1 mais sont aussi conçus pour être réutilisables et sont également fournis à l'utilisateur, qui peut également en décrire de nouveaux. Les composants du système et les attaquants sont représentés par des automates. Il communiquent entre-eux en lisant et en écrivant sur des canaux, en accord avec la topologie

9. Les modèles utilisés peuvent être trouvés à l'adresse suivante : https://maxime.puys.name/files/phdThesisAssets.tar.bz2

du système. Comme détaillé en section 2.2, les messages échangés sont des commandes de lecture et écriture sur des variables. Les clients envoient des requêtes auxquelles les serveurs répondent, en modifiant les variables du procédé dans le cas des écritures. Les attaquants peuvent soit se présenter sur un canal (en MITM), soit sous forme d'un client ou d'un serveur qui aurait été corrompu. En accord avec la phase 1, ils pourront agir de différentes façons sur le système.

On va alors combiner les modèles du système et des attaquants (6) afin de vérifier si les propriétés peuvent être mises en défaut par les attaquants et, le cas échéant, de fournir une trace d'exécution menant à l'attaque. Pour cela plusieurs méthodes peuvent être envisagées, par exemple le model-checking. Il se peut enfin que l'analyse soit inconclusive (par exemple parce que le modèle à analyser est trop important). Dans le cadre de cette thèse, nous avons choisi d'implémenter cette approche avec le model-checker UPPAAL. L'approche peut cependant être étendue à d'autres méthodes de vérification. On commence par expliquer en section 5.4.1.1 comment sont modélisés les clients et les serveurs. Ensuite, en section 5.4.1.2, on décrit la modélisation des attaquants. Quatre attaquants sont donnés en exemple. Enfin, la section 5.4.1.3 explique la modélisation des propriétés de sûreté à vérifier.

5.4.1.1. Modélisation des clients et des serveurs

Comme décrit ci-dessus, plusieurs processus peuvent être réutilisés d'une modélisation à l'autre. Il s'agit par exemple des processus implémentant les clients et les serveurs pour chaque protocole (qui encapsulent et décapsulent les requêtes et les réponses échangées). De façon similaire, en abstrayant les primitives cryptographiques utilisées, un automate décrivant la sécurité des messages peut être commun à tous les protocoles. Ainsi dans l'implémentation en UPPAAL, les automates gérant le comportement des clients et des serveurs sont respectivement nommés *BehaviorClient* et *BehaviorServer* tandis que l'automate gérant la sécurité des protocoles est nommé *SecurityLayer*. En UPPAAL, les canaux sont utilisés comme des signaux et non pour transférer des données. Ainsi celles-ci sont accédées via des variables globales. Cela s'applique par exemple aux clés cryptographiques ou aux messages échangés par les clients et les serveurs. En particulier, dans la suite de cette section le client et le serveur possèdent deux clés partagées : K_{CS} pour le chiffrement et $K_{Sig_{CS}}$ pour les signatures.

Pour envoyer un message, l'automate BehaviorClient envoie le contenu applicatif (les commandes) à l'automate Client qui formate le message selon le protocole implémenté. Ensuite, dans le cas d'un protocole sécurisé, le message est transféré à l'automate SecurityLayer qui le renvoie signé et/ou chiffré avec les clés cryptographiques du client. Enfin, le message est envoyé sur le canal en direction du serveur. A la réception de la réponse, le message est déchiffré et sa signature vérifiée par l'automate SecurityLayer si le protocole est sécurisé. Enfin, le contenu applicatif est transféré à BehaviorClient pour mettre à jour le comportement en fonction de la réponse du serveur. Ces interactions sont résumées dans la figure 5.7.

L'automate Server attend de recevoir un message sur le canal du client. Une fois reçu, dans le cas d'un protocole sécurisé, le message est transféré à l'automate *SecurityLayer* qui vérifie la signature et/ou déchiffre le message avec les clés cryptographiques du serveur. Le message est ensuite transféré à l'automate BehaviorServer qui, en fonction

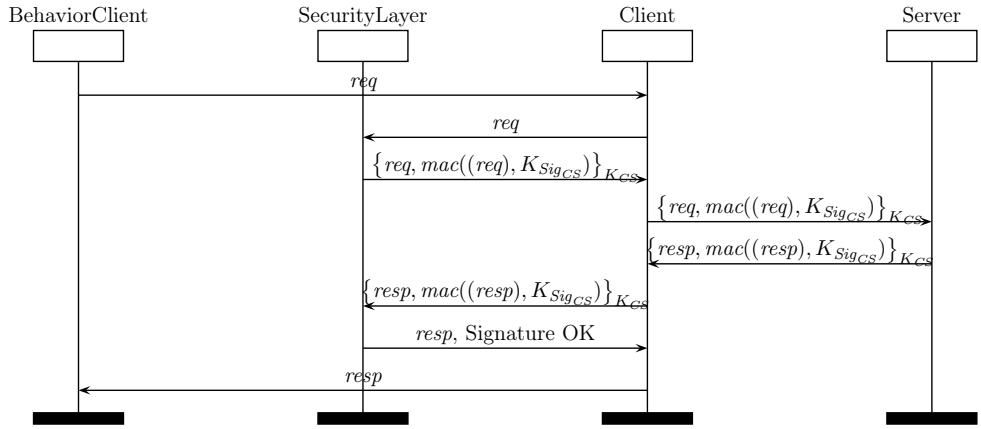


FIGURE 5.7. – Diagramme de séquence du client

du type de commande (lecture ou écriture), change la valeur d'une variable écrite ou lit sa valeur courante. L'automate Server crée alors une réponse pour le client (confirmation de l'écriture ou valeur lue). Dans le cas d'un protocole sécurisé, le message est à nouveau transféré à l'automate SecurityLayer pour être signé et/ou chiffré et enfin envoyé sur le canal du client. Ces interactions sont résumées dans la figure 5.8.

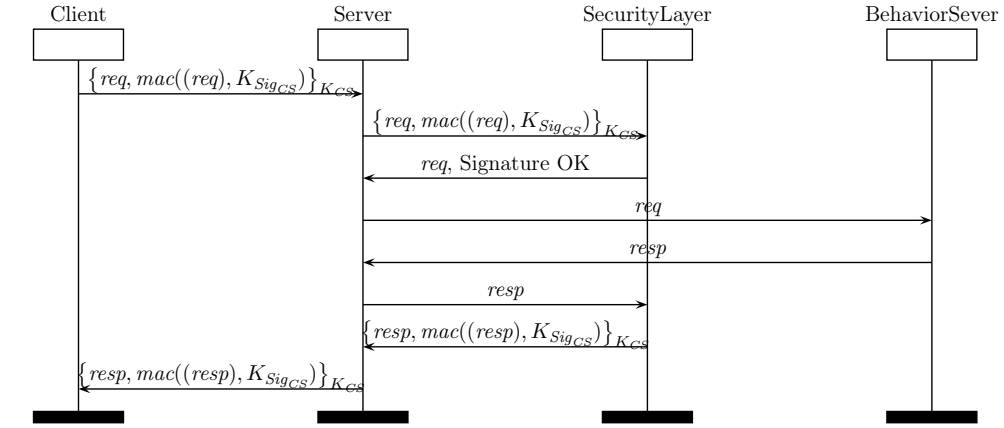


FIGURE 5.8. – Diagramme de séquence du serveur

5.4.1.2. Modélisation des attaquants

Nous proposons, à titre d'exemple, quatre attaquants avec différentes capacités, chacun modélisé par un automate. Ces attaquants sont proposés à titre d'exemple et afin de montrer plusieurs résultats possibles. L'utilisateur est libre de spécifier d'autres attaquants en fonction des résultats des analyses de risques (en particulier de la phase

1). L'attaquant A_1, étant basé sur le modèle Dolev-Yao, il peut intercepter les messages envoyés sur le réseau, les bloquer, les modifier, les rejouer ou encore en forger de nouveau en fonction de ses connaissances. Dans la figure 5.9a, l'exécution de l'automate démarre dans l'état A_1 où l'attaquant peut choisir l'action qu'il souhaite effectuer. Ces actions sont directement liées aux vecteurs d'attaques (*Lire*, *Mod*, etc.) introduits dans la phase 1. Les états possibles de l'automate de l'attaquant A_1 sont :

— *Intercept* qui lui permet d'intercepter un message *msg* envoyé par un client ou un serveur sur un canal *chan* ;

— *Send* qui lui permet d'envoyer un message *msg* à un client ou un serveur sur un canal *chan* ;

— *Copy* qui lui permet de mémoriser un message *msg* dans sa base de connaissances $\mathcal{K}_{A_1}$;

— *Keys* qui lui permet de récupérer des clés cryptographiques depuis sa base de connaissances $\mathcal{K}_{A_1}$;

— *Security* qui lui permet de réaliser des opérations cryptographiques en accord avec les clés mémorisées dans sa base de connaissances $\mathcal{K}_{A_1}$;

— *Forge* qui lui permet de créer un nouveau message *msg* à partir de sa base de connaissances $\mathcal{K}_{A_1}$;

— *Replay* qui lui permet de rejouer un message *msg* mémorisé dans sa base de connaissance $\mathcal{K}_{A_1}$;

— *Modify* qui lui permet de modifier un message – ou une partie d'un message – *msg* à partir de sa base de connaissances $\mathcal{K}_{A_1}$;

Les trois autres attaquants sont des sous-ensembles de A_1 qui sont restreints à une capacité (par exemple forger un message). L'attaquant A_2 (présenté en figure 5.9b) est un sous-ensemble de l'attaquant A_1 qui est restreint à la modification de messages – ou de parties de messages. Pour plus de réalisme, il pourrait par exemple être limité à ne modifier que les variables ou les valeurs échangées sans pouvoir transformer une requête de lecture en écriture et vice versa. Cela pourrait représenter un attaquant tentant d'éviter des attaques grossières afin de rester le plus discret possible. L'attaquant A_3 (présenté en figure 5.9c) est un sous-ensemble de l'attaquant A_1 qui est restreint à la création de nouveaux messages à partir de sa base de connaissances. Cela peut représenter un attaquant « aveugle » qui peut envoyer des messages sur le réseau mais qui ne peut pas en intercepter (par exemple parce qu'il se trouve sur un canal isolé). Enfin, l'attaquant A_4 (présenté en figure 5.9d) est un sous-ensemble de l'attaquant A_4 qui est restreint au rejeu de messages qu'il aura précédemment intercepté et mémorisé dans sa base de connaissances. Cela peut représenter un attaquant en position de MITM, voyant passer des messages chiffrés et n'ayant pas les clés cryptographiques correspondantes.

On remarque que le comportement des attaquants n'est pas borné (de façon similaire aux problématiques évoquées en section 4.2). Pour permettre la terminaison de l'analyse, nous avons borné le nombre maximal d'actions qu'ils peuvent effectuer dans une étape d'une attaque, cette borne étant configurable. La table 5.4 résume les capacités des quatre attaquants présentés avec le symbole ✓ indiquant que l'attaquant peut effectuer

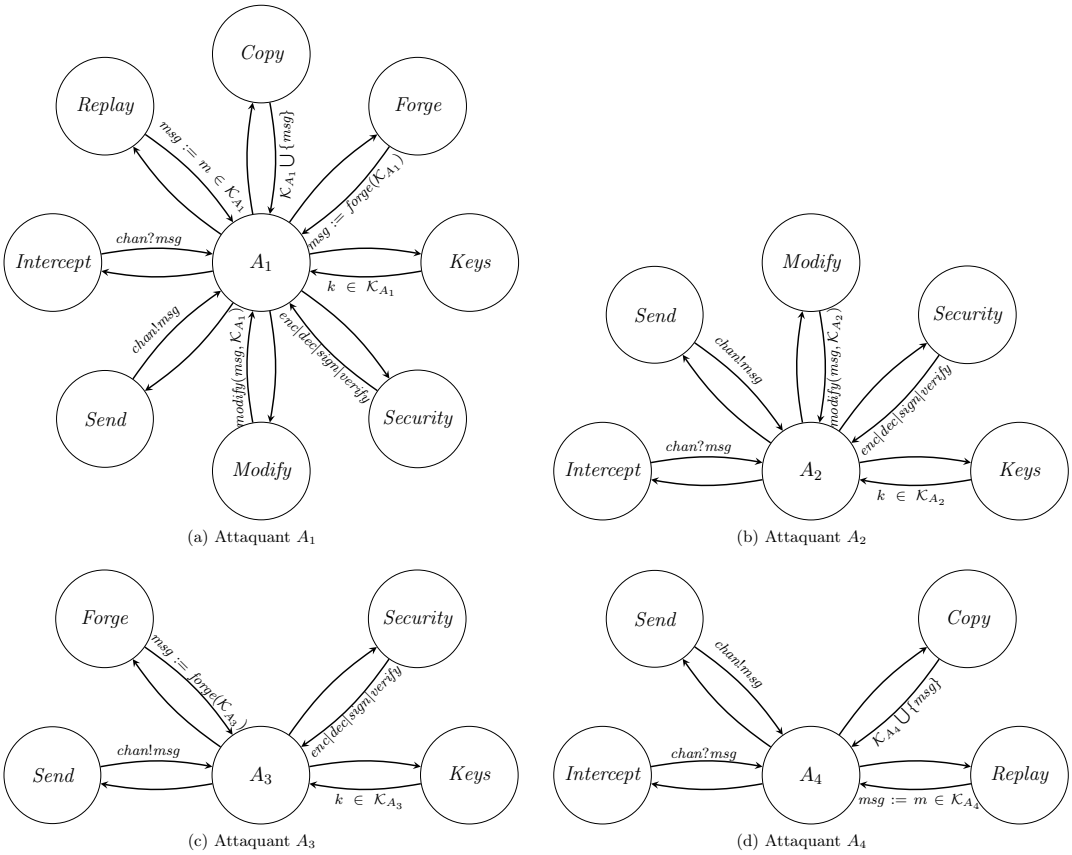


FIGURE 5.9. – Attaquants proposés

l'action. *Replay* et *Modify* peuvent être vus comme des cas particulier de *Forge*, mais nécessitent que le message rejoué ou modifié ait été précédemment intercepté. A l'opposé, *Forge* permet à l'attaquant d'envoyer un message à tout instant.

Attaquant	Modify	Forge	Replay
A_1	✓	✓	✓
A_2	✓	✗	✗
A_3	✗	✓	✗
A_4	✗	✗	✓

TABLE 5.4. – Résumé des capacités des attaquants proposés

5.4.1.3. Modélisation et vérification des propriétés de sûreté

Pour spécifier les propriétés de sûreté Φ que l'attaquant cherche à violer, UPPAAL utilise une version simplifiée de la logique CTL qui s'exprime via la syntaxe suivante.

$$\Phi ::= A\Box\Phi | E \diamond \Phi | E\Box\Phi | A \diamond \Phi | \Phi \rightarrow \Phi | \neg\Phi$$

$A\Box\Phi$ signifie que Φ doit être vrai sur tous les chemins dans tous les états accessibles. $A \diamond \Phi$ signifie que Φ doit devenir vrai sur tous les chemins. $E\Box\Phi$ signifie qu'il existe un chemin dans lequel Φ est vrai dans tous les états accessible. $E \diamond \Phi$ signifie qu'il existe un chemin dans lequel Φ doit devenir vrai. Les symboles $\rightarrow$ et $\neg$ dénotent respectivement les opérateurs implication et négation de la logique propositionnelle. Dans le cas de l'approche A^{2}SPICS, nous utilisons uniquement sur des propriétés de la forme $A\Box\Phi$. Les modèles du système et des attaquants sont alors combinés et les processus sont exécutés en parallèle en se synchronisant sur les canaux afin de vérifier les propriétés.

5.4.2. Étude de cas : une chaîne de mise en bouteille

Dans cette section, on modélise l'exemple présenté en section 5.2.2 avec les paramètres l'approche A^{2}SPICS. On considérera deux topologies et plusieurs attaquants afin de montrer que de simples variations des paramètres peuvent changer les résultats obtenus.

5.4.2.1. Comportements

Comme décrit en section 5.2.2, nous nous intéressons à une usine de remplissage de bouteilles. Les bouteilles vides avancent sur un tapis roulant et un capteur détecte lorsqu'elles sont positionnées sous une vanne. Le tapis roulant s'arrête et la vanne s'ouvre pour remplir la bouteille de liquide. Un second capteur détecte lorsque la bouteille est pleine. La vanne se ferme et le tapis roulant redémarre. Enfin, un client peut démarrer et arrêter l'ensemble du procédé.

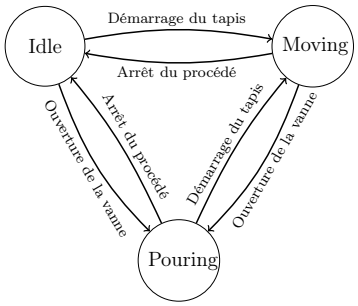
Comportement du procédé : Dans cet exemple, le procédé est composé de cinq variables booléennes :

$$\mathcal{V}_{\mathcal{P}} = \{motor, nozzle, levelHit, bottleInPlace, processRun\}$$

Elles dénotent respectivement le tapis roulant (*motor*), la vanne (*nozzle*), le capteur de niveau de liquide (*levelHit*), le capteur de position des bouteilles (*bottleInPlace*) et l'interrupteur marche/arrêt du procédé (*processRun*). La figure 5.10a montre un automate décrivant le comportement du procédé et la table 5.10b détaille les transitions de l'automate. Parmi les trois états de l'automate, *Idle* signifie que le procédé est à l'arrêt, *Moving* que le tapis roulant avance et *Pouring* qu'une bouteille se remplit. Chaque transition est étiquetée par une garde et une sortie.

Comportement du client : Dans notre exemple, le client se contentera de démarrer et d'arrêter le procédé, comme montré en figure 5.11. L'ensemble des variables auquel il accède est donc : $\mathcal{V}_c = \{processRun\}$.

Propriétés à garantir : Les propriétés que le système doit garantir sont un sous-ensemble des propriétés résultant d'une analyse de risques en sûreté, considérées comme critiques. Dans le cadre de cet exemple, nous définissons les propriétés Φ_1, Φ_2, et Φ_3, modélisées en logique CTL de façon suivante :



(a) Automate du comportement du procédé

État courant	État suivant	Garde	Actions
Idle	Moving	$processRun = true \wedge bottleInPlace = false$	$motor := true$
Idle	Pouring	$processRun = true \wedge bottleInPlace = true$	$nozzle := true$
Moving	Pouring	$bottleInPlace = true$	$motor := false \wedge nozzle := true$
Pouring	Moving	$levelHit = true$	$motor := true \wedge nozzle := false$
Moving	Idle	$processRun = false$	$motor := false \wedge nozzle := false$
Pouring	Idle	$processRun = false$	$motor := false \wedge nozzle := false$

(b) Détail des transitions

FIGURE 5.10. – Comportement du procédé

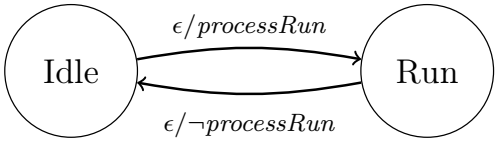


FIGURE 5.11. – Comportement du client

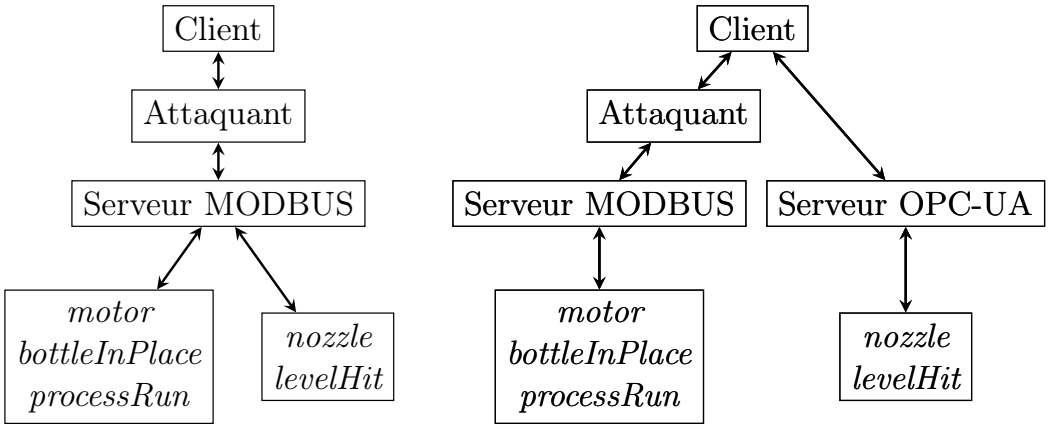
— Φ_1 : La vanne ne s'ouvre que lorsqu'une bouteille est en position (i.e. : à chaque instant et sur chaque chemin, *nozzle* ne vaut jamais *true* si *bottleInPlace* vaut *false*).
$A\Box \neg(nozzle = true \ and \ bottleInPlace = false)$

— Φ_2 : Le moteur ne démarre que lorsque la bouteille est pleine (i.e. : à chaque instant et sur chaque chemin, *motor* ne vaut jamais *true* si *levelHit* vaut *false*).
$A\Box \neg(motor = true \ and \ levelHit = false)$

— Φ_3 : La vanne ne s'ouvre que lorsque le moteur est arrêté (i.e. : à chaque instant et sur chaque chemin, *nozzle* et *motor* ne valent jamais *true* en même temps).
$A\Box \neg(nozzle = true \ and \ motor = true)$

5.4.2.2. Topologies

On considère deux topologies réseau possibles T_1 et T_2. Dans la topologie T_1, le tapis roulant et la vanne sont contrôlés par un même serveur qui communique avec le client via le protocole MODBUS. La topologie T_1 est représentée en figure 5.12a. Dans la topologie T_2, le tapis roulant et la vanne sont contrôlés chacun par un serveur. Le serveur s_{MODBUS} contrôle le tapis roulant, le capteur de position et l'interrupteur marche/arrêt du procédé. Le serveur $s_{\text{OPC-UA}}$ contrôle la vanne et le capteur de niveau. Comme décrit dans les chapitre précédent, le protocole OPC-UA propose trois modes de sécurité : None, Sign et SignEncrypt. Le mode None n'apporte aucune sécurité, tandis que les modes Sign et SignEncrypt ajoutent des signatures cryptographiques et du chiffrement pour le dernier. On suppose que le mode SignEncrypt est utilisé dans cette topologie, empêchant l'attaquant d'interférer avec le canal entre le client c et le serveur $s_{\text{OPC-UA}}$. La topologie T_2 est représentée en figure 5.12b. Plusieurs autres topologies peuvent être considérées avec par exemple l'attaquant prenant le contrôle du client ou d'un serveur. On peut même envisager qu'il se soit inséré directement entre le serveur et le procédé



(a) Topologie 1 (b) Topologie 2

FIGURE 5.12. – Topologies réseau modélisées

en attaquant un bus de terrain. Pour plus de détails sur d'autres topologies possibles, le lecteur peut se référer à l'annexe A.1.

5.4.3. Résultats obtenus avec UPPAAL

Dans l'idée globale de l'approche A^{2}SPICS, les attaquants considérés devraient résulter de l'analyse de risques en phase 1. Ici, afin de présenter divers résultats possibles de l'analyse, nous testons chaque propriété avec l'ensemble des attaquants possibles. La table 5.5 présente les résultats fournis par UPPAAL. Le symbole ✓ signifie qu'une attaque a été trouvée, le symbole ✗ signifie que la propriété est vérifiée et enfin ● signifie que UPPAAL n'a pas pu conclure. Dans nos expérimentations, tous les cas inconclusifs sont dus à un dépassement de la mémoire disponible. Nos expérimentations ont été menées sur un processeur Intel(R) Core(TM) i5-4590 CPU @ 3.30GHz avec 16GB de RAM.

Topologies	**Propriétés**	A_1	A_2	A_3	A_4
T_1	Φ_1	✓	✓	✓	✗
	Φ_2	✓	✓	✓	✗
	Φ_3	✓	✓	✓	✗
T_2	Φ_1	●	●	✗	✗
	Φ_2	✓	✓	✓	✗
	Φ_3	✓	✓	✓	✗

TABLE 5.5. – Résultats obtenus par UPPAAL

En principe, aucun des quatre attaquants ne peut violer la propriété Φ_1 dans la topologie 2. En effet, pour cela il faut avoir à la fois la variable *nozzle* à *true* et la variable *bottleInPlace* à *false*. Or, lorsque la variable *bottleInPlace* passe à *false*, la variable *nozzle* passe automatiquement à *false* dû au comportement du serveur (présenté en figure 5.10a). L'attaquant doit donc obligatoirement forcer la variable *nozzle* à *true*

pour violer la propriété, ce qui n'est pas possible puisque celle-ci est contrôlée par le serveur OPC-UA (plus de détails sur ce résultat en section 5.6.3). De façon similaire, l'attaquant A_4 ne peut violer aucune propriété du fait qu'il se contente de rejouer des messages transmis entre les clients et serveurs. Or le client n'envoie des messages que pour démarrer et arrêter le processus, ce qui n'est pas suffisant pour réaliser les attaques Φ_1, Φ_2 et Φ_3.

Observations sur les résultats : D'après les tables 5.5 et 5.6, l'attaquant A_2 obtient les mêmes résultats que l'attaquant A_1 (Dolev-Yao) en moins de temps. L'attaquant A_3 requiert un peu plus de temps de calcul mais est en mesure de conclure dans le cas de la propriété Φ_1 dans la topologie 2 alors que les attaquants A_1 et A_2 n'y parviennent pas. Cela montre que les attaquants implémentés dans la logique de certains outils tels que l'intrus Dolev-Yao sont souvent très complexes (en termes de règles de déduction et de capacités) et un sous-ensemble de ces capacités peut suffir à trouver des attaques (en particulier ici, plusieurs capacités de Dolev-Yao suffisent chacune seule). L'attaquant A_4 obtient des temps de calcul plus longs que les autres. Cela peut paraître étonnant étant donné que c'est le plus simple de nos attaquants Une explication possible est que comme tous ces résultats sont une absence d'attaque, UPPAAL doit à chaque fois explorer tous les chemins possibles ce qui peut prendre plus de temps que de trouver une attaque.

Topologies	**Propriétés**	A_1	A_2	A_3	A_4
T_1	Φ_1	0.43 s	0.07 s	1.05 s	0.84 s
	Φ_2	0.52 s	0.10 s	0.69 s	0.35 s
	Φ_3	0.47 s	0.04 s	0.37 s	0.42 s
T_2	Φ_1	Mémoire épuisée		601 s	31.55 s
	Φ_2	0.66 s	0.23 s	2.17 s	35.20 s
	Φ_3	0.78 s	0.21 s	2.35 s	34.85 s

TABLE 5.6. – Temps de vérification pour chaque propriété

Cette étude de cas considère un exemple très simple mais montre la faisabilité de l'approche et permet déjà une comparaison sommaire des attaquants. La section 5.6.3 discutera des limites des deux phases de l'approche. Une autre piste envisagée a été d'utiliser des outils de vérification de protocoles cryptographiques pour vérifier le modèle, ces derniers étant optimisés pour des analyses en présence d'attaquants. Cette possibilité est discutée en section 5.5.

5.5. Utilisation d'outils de vérification de protocoles cryptographiques

Historiquement, les outils de vérification comme UPPAAL ont été utilisés pour vérifier des propriétés de sûreté de fonctionnement (ou de vivacité bornée). A notre connaissance, ce n'est qu'en 1998 que Lowe présente Casper/FDR [Lowe, 1998], un outil permettant de vérifier des protocoles cryptographiques en présence de l'attaquant Dolev-Yao à l'aide du model-checker FDR. La particularité de cet outil réside dans le fait que la logique

de l'attaquant (ses capacités et son système de déduction) est incluse dans le modèle FDR par Casper/FDR et non à modéliser par l'utilisateur. Ces travaux ont mené à la définition d'une nouvelle classe d'outils, les outils de vérification de protocoles cryptographiques, taillés pour des propriétés de sécurité comme le secret et l'authentification en présence d'attaquants. Le fonctionnement de ces outils a été décrit plus en détail dans la section 4.2 du chapitre 4. Afin d'implémenter l'approche A²SPICS, on peut alors se demander quelle classe d'outils est la plus adaptée. Il parait en effet tentant de profiter du fait que les outils de vérification de protocoles implémentent nativement le rôle de l'attaquant. En parallèle des travaux d'implémentation avec UPPAAL, nous avons donc mené des expérimentations avec de tels outils.

Une première limite des outils de vérification de protocoles cryptographiques vient de l'expression des propriétés à vérifier. Dans le cadre de l'approche A²SPICS, les propriétés sont exprimées sous forme de prédicats logiques sur les valeurs des variables des serveurs. Or, la plupart des outils sont intrinsèquement dédiés à certaines propriétés de sécurité comme la confidentialité d'un terme ou l'authentification des agents. A notre connaissance, seuls ProVerif et Tamarin permettent d'exprimer nativement des propriétés sous forme de formules logiques. Dans le cadre du projet AVANTSSAR, les auteurs de la suite AVISPA ont également permis l'expression de telles propriétés via le compilateur ASLAN++. L'application de cette suite d'outils à des systèmes industriels a notamment été montrée par Rocchetto et Tippenhauer en 2017 [Rocchetto and Tippenhauer, 2017]. Ils se heurtent néanmoins à la difficulté de modifier le comportement de l'intrus Dolev-Yao dans l'outil CL-Atse.

Une seconde limite des outils de vérification de protocoles cryptographiques vient du fait que, pour faire face à l'explosion combinatoire inhérente aux problèmes de vérification, ces outils font de nombreuses abstractions. On peut alors se demander si ces abstractions, valides pour la vérification de protocoles cryptographiques, sont utiles ou non, voir limitantes, pour vérifier des propriétés de sûreté de fonctionnement. Dans cette section. nous étudions notamment la possibilité d'implémenter l'approche A²SPICS à l'aide de l'outil ProVerif.

5.5.1. Limites de ProVerif pour l'approche A²SPICS

ProVerif[10] [Blanchet, 2001, Blanchet et al., 2017] a été principalement développé en 2001 et est aujourd'hui toujours maintenu par Bruno Blanchet. C'est un outil de vérification de protocoles cryptographiques qui analyse un nombre infini de sessions du protocole en parallèle en présence d'un intrus Dolev-Yao. D'abord uniquement modélisés en clauses de Horn [Horn, 1951], les protocoles peuvent maintenant être écrits dans un sous-ensemble du π-calcul [Milner et al., 1992] (alors retransformés en clauses de Horn par l'outil). Il permet notamment l'expression de fonctions, types et théories équationnelles utilisables dans la modélisation, en particulier diverses primitives cryptographiques. Les propriétés sont exprimées sous forme d'implication d'événements pouvant être levés dans le modèle. Ces événements sont alors ajoutés à la trace sous forme d'étiquettes et il est possible de raisonner sur leur chronologie (par exemple vérifier qu'un message a été envoyé avant d'être reçu). ProVerif se base sur des techniques

10. http://prosecco.gforge.inria.fr/personal/bblanche/proverif/

de sur-approximation. Il peut donc prouver avec certitude l'absence d'attaques mais les contre-exemples qu'il trouve peuvent être des faux positifs. Dans ce cas, l'outil tente de reconstruire une trace menant à l'attaque.

Cependant, il apparaît que certaines abstractions faites par ProVerif posent problème pour l'utiliser dans le cadre de l'approche A^{2}SPICS. Pour rappel, on souhaite modéliser des processus qui bouclent pour envoyer et recevoir des commandes et on a également besoin de représenter des états (par exemple l'automate du comportement du procédé). En particulier dans ProVerif : (i) aucun mécanisme ne permet de partager de la mémoire entre des processus, (ii) il n'est pas possible qu'un processus soit récursif, et (iii) les messages transmis sur des canaux ou insérés dans des tables (une autre structure de données) ne peuvent être consommés (ils peuvent donc être utilisés sans limite). L'absence des deux premières fonctionnalité vient essentiellement du fait qu'elles ne sont pas utiles pour vérifier des protocoles cryptographiques. Le troisième point est directement lié à la modélisation des faits en clauses de Horn. En effet, contrairement à la logique linéaire introduite par Girard en 1987 [Girard, 1987], en clauses de Horn, les faits ne sont pas consommés. L'ensemble des faits augmente donc de façon monotone. Cela signifie qu'un message envoyé une fois peut être reçu un nombre infini de fois sans que l'intrus ait besoin de le rejouer (idem dans le cas des tables). Dans le cas de l'approche A^{2}SPICS, cela signifie que si l'on sauvegarde l'état global du système, le dernier état ne remplacera pas les anciens qui pourront être chargés à la place. Il est donc extrêmement compliqué de maintenir un état global du système à l'aide de canaux ou de tables.

De plus, les protocoles de transport comme MODBUS et OPC-UA sont un enchaînement de requêtes et réponses, contrairement aux protocoles de handshake qui sont une séquence fixée d'actions. Cependant aucun mécanisme de boucle ou récursion n'est possible sur les processus en ProVerif. Il est possible de le simuler avec l'opérateur de réplication de processus (nommé « ! ») mais cet opérateur génère une copie du processus en parallèle du premier sans contrôle de l'ordre dans lequel ils s'exécutent et toujours avec le problème des canaux et des tables pour qu'ils se passent de l'information. Dans le reste de cette section, nous proposons des constructions en π-calcul afin dépasser ses limitations.

5.5.2. Modéliser un état mémoire avec ProVerif

Nous proposons une structure de données en ProVerif qui permet de mémoriser les valeurs des variables afin de vérifier nos propriétés qui sont des prédicats sur les valeurs des variables du système. Nous définissons un type nommé array représentant des couples de variables et valeurs. Nous définissons également un fonction *store* et un prédicat *bel* afin de les manipuler. On peut alors définir une structure de données fonctionnant comme une liste chaînée : $store(var, val, mem)$ affecte une nouvelle valeur val à la variable var dans la mémoire mem. La liste est initialisée grâce à une constance nommé *empty*, désignant la mémoire vide. Le prédicat *bel* (pour *belong*) est utilisé pour vérifier la dernière valeur connue d'une variable. Ainsi, la mémoire est une liste imbriquée d'appels à *store*. C'est une façon courante de représenter et d'axiomatiser des tableaux dans les outils de vérification.

Par exemple, $store(x, true, store(y, true, store(x, false, empty)))$ désigne l'état dans lequel x et y valent tous les deux *true*. Même si x a pu avoir comme valeur *false* à un

certain instant, le dernier appel à *store* (le plus à l'extérieur) a remplacé sa valeur par *true*. Les axiomes du prédicat *bel* sont donnés en listing 5.1 où la première clause sera vrai si la variable *var* a été mise à *val* dans le dernier appel à *store* de l'état *mem*. Dans le cas contraire, la seconde clause parcourra les appels à *store* précédents de *mem*. On peut remarquer que cette solution a le défaut d'obliger à énumérer les valeurs possibles des variables, c'est un problème courant dans les modèle en ProVerif.

```
pred bel(bitstring, bool, array).
clauses
   forall var, val, mem; bel(var, val, store(var, val, mem));
   forall var, nextVar, val, nextVal, mem; (var ≠ nextVar)
   ∧ bel(var, val, mem) → bel(var, val, store(nextVar, nextVal, mem)).
```

Listing 5.1 – Équations pour gérer la mémoire

La mémoire est manipulée par un processus appelé *mema*, montré dans le listing 5.2. Ce processus prend en paramètre un état mémoire *state* et il requiert un compteur et un canal pour des raisons de synchronisation décrites ci-après. Il commence par récupérer une action à effectuer (lecture ou écriture) dans la table *tActions*. Cette action est un triplet (cmd, var, val) avec *cmd* étant une lecture ou écriture, et var, val une variable et une valeur (dans le cas d'une écriture). Dans le cas d'une lecture, il récupérera la dernière valeur connue de *var* par l'intermédiaire du prédicat *bel*. Dans le cas d'une écriture, le processus *mema* utilisera la fonction *store* pour modifier la valeur de *var* dans une nouvelle mémoire *nstate* qu'il utilisera lors de sa prochaine exécution.

```
let mema(mcount: natP, c_son_m: channel, state: array) =
    get tActions(com, var, val, resp_serv, =mcount) in
    if (com=read) then
        if bel(var, true, state) then
            out(resp_serv, (true, state))
        else
            out(resp_serv, (false, state))
    else
        let nstate=store(var, val, state) in
        insert tObs(var, val, state);
        out(resp_serv, (val, state));
        out(c_son_m, (S(mcount), nstate)).
```

Listing 5.2 – Processus pour gérer la mémoire

Les actions effectuées sont enregistrées dans la table *tObs* avec l'état courant de la mémoire. Cette solution permet de vérifier les propriétés sur les valeurs des variables via une processus observateur. Par exemple le processus observateur du listing 5.3 vérifie qu'une variable *var* n'a pas été mise à la même valeur *val* que sa valeur actuelle. Cette propriété est par exemple utile dans le cas du sectionneur électrique présenté en annexe A.2.

```
let obs =
    get tObs(var, val, mem) in if (bel(var, val, mem)) then
    event badState.
```

Listing 5.3 – Processus observateur

5.5.2.1. Processus récursifs

Comme nous nous intéressons à des protocoles qui sont une séquence (potentiellement infinie) de commandes de lecture et écriture, nous devons donc modéliser des processus qui peuvent accepter un nombre infini de commandes. Comme ProVerif ne propose pas de mécanisme de boucles, il est possible de reproduire ce comportement avec l'opérateur de réplication. Cependant, comme toutes les copies du processus s'exécutent en parallèle, leurs états et leurs messages peuvent se mélanger. Nous proposons donc que chaque processus n'utilise que des canaux frais afin d'éviter tout « mélange ». De plus, pour maintenir un ordre dans l'exécution des processus, nous leur associons un compteur, modélisé sous forme d'un entier de Peano[11]. Ainsi, nous pouvons expliquer plus en détail les paramètres du processus *mema*. Pour une certaine exécution du processus *mema*, *mcount* (resp. *state*) représentent le compteur (resp. l'état) qui a été généré par l'exécution précédente de *mema*. La commande récupérée dans la table *tActions* est par ailleurs estampillée par le compteur *mcount* afin que chaque exécution de *mema* soit associée avec la bonne commande envoyée. Cela implique que ce compteur soit aussi global aux clients, aux serveurs afin qu'ils avancent pas à pas en restant synchronisés. Chaque exécution du processus est lancée avec un canal sur lequel l'exécution suivant viendra lire la compteur et l'état mis à jour. Le listing 5.4 montre ce processus de réplication.

```
mema(Z, c1i, mm) |
out(ca, c1i)   |
(!( new c2i: channel ;
    in(ca, cmi: channel);
    in(cmi, (mcount: natP, state: array));
    (out(ca, c2i) | mema(mcount, c2i, state))
))
```

Listing 5.4 – Processus de réplication

5.5.2.2. Transfert de commandes du client au serveur

Dû à la représentation des messages en clauses de Horn par ProVerif, un message envoyé peut être reçu sans limite sans qu'un attaquant n'ait besoin de le rejouer. Dans les protocoles de transport, les messages étant associés à un numéro de séquence, nous leur associons dans la modélisation en ProVerif le compteur du client. Par ailleurs, toujours pour éviter que les messages ne soient reçus plus de fois qu'ils ne sont envoyés, chaque message du client au serveur est transmis sur un canal frais. Pour cela, chaque message est aussi accompagné du canal qui sera utilisé pour le message suivant. De plus les requêtes et les réponses sont envoyées sur des canaux distincts. Le même système de canaux est utilisé entre le serveur et *mema*. Le listing 5.5 décrit le processus du serveur, qui utilise le même procédé de réplication que *mema*.

```
let server(c_to_s, s_to_c, c_son_s: channel, cpts: natP) =
    in(c_to_s, (=cpts, c_next, c_i_next, resp_cli: channel,     \
```

11. Un constructeur Z représente le nombre de base et une fonction S permet de construire le successeur d'un nombre

```
        (com: cmd, var: bitstring, val: bool)));
    new mem_to_s: channel; new nounce: bitstring;

    insert tActions(com, var, val, mem_to_s, cpts);
    in(mem_to_s, (x: bool, y: array));

    out(resp_cli, (x, y));
    out(c_son_s, (S(cpts), c_next, c_i_next)).
```

Listing 5.5 – Processus des serveurs

5.5.2.3. Modélisation d'attaquant

L'un des buts de l'approche A²SPICS est de permettre de tester un système face à des attaquants personnalisés. Ainsi, il est donc nécessaire de pouvoir restreindre les capacités de l'intrus Dolev-Yao de ProVerif sans perdre pour autant ses capacités de déduction et d'ordonnancement des processus. De façon similaire à l'implémentation avec UPPAAL en section 5.4, nous isolons les capacités de Dolev-Yao en des intrus distincts afin de voir lesquelles sont nécessaires à l'attaque. En général, les outils de vérification de protocoles cryptographiques supposent que tous les agents communiquent sur un unique canal accessible à l'intrus Dolev-Yao. Ainsi, pour éviter que l'intrus Dolev-Yao implémenté dans ProVerif ne contrôle totalement les canaux, nous choisissons de les rendre privés. L'intrus ne peut alors plus effectuer aucune action (il garde néanmoins ses capacités de déduction). On lui redonne alors des accès restreints aux canaux privés via des oracles qui agissent alors comme des « proxy », lui permettant de ne faire que certaines actions (par exemple un oracle qui ne peut que modifier les messages).

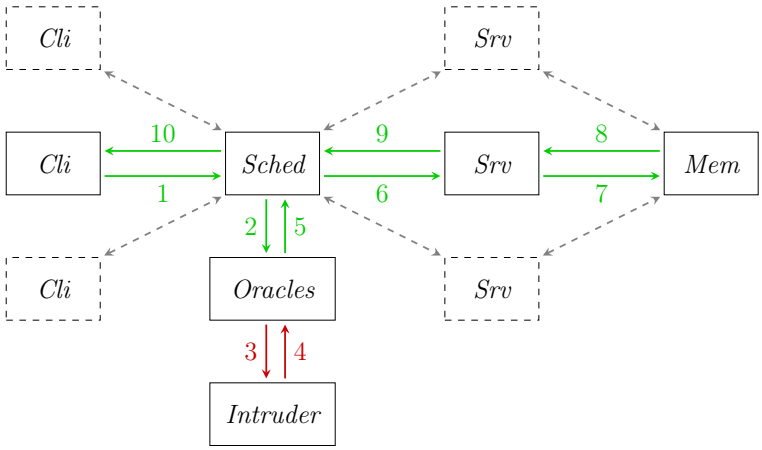


FIGURE 5.13. – Topologie réseau avec intrus en ProVerif

La figure 5.13 montre une topologie réseau permettant cette limitation des capacités de l'intrus Dolev-Yao, avec en vert les canaux privés et en rouge les canaux publics, accessibles à Dolev-Yao. Le client commence par envoyer une requête au serveur (1). Pour que l'intrus puisse potentiellement l'intercepter, elle passe par un scheduler qui l'envoie aux oracles (2, 3). Ce scheduler est nécessaire afin que les oracles n'accèdent pas directement au canal entre les clients et les serveurs. Le scheduler attend donc le

message potentiellement attaqué par l'intrus (4,5) et le fait alors suivre au serveur avec lequel le client parle (6) qui lui communique avec la mémoire (7, 8). Le serveur renvoie la réponse vers le client (9) via le scheduler. La réponse est également transmise aux oracles (non représenté sur la figure) pour que l'intrus puisse attaquer la réponse. Enfin, la réponse est transmise au client (10).

Cette configuration permet à l'intrus d'intercepter des messages entre le client et le serveur. Elle complique cependant la définition de la topologie car un scheduler est nécessaire sur chaque canal où un attaquant est présent. Cependant, l'intrus peut aussi agir de lui-même (forger ou rejouer un message). Dans ce cas, seules les flèches (4, 5, 6, 7, 8, 9) doivent être considérées. Le listing 5.6 montre par exemple deux oracles permettant respectivement d'écouter sur les canaux privés [12] et d'inverser la valeur booléenne d'une variable contenue dans une requête ou une réponse.

```
let oracle_wiretap =
   in (c_input, (command: cmd, variable: bitstring, value: bool));
   out (c_intruder, (command, variable, value));
   out (c_output, (command, variable, value)).
let oracle_modifboolvalue =
    in (c_input, (command: cmd, variable: bitstring, value: bool));
    let newValue = if value = false then true else false in
    out (c_output, (command, variable, newValue)).
```

Listing 5.6 – Exemples d'oracles

5.5.2.4. Résultats de l'étude avec ProVerif

Cet encodage de la mémoire est à notre connaissance l'un des premiers à permettre de stocker des états dans ProVerif sans passer par une modification du code source de l'outil [Arapinis et al., 2014]. Cependant, il présente plusieurs limitations. Premièrement, la complexité du système de canaux frais rend très difficile la modélisation de topologies avec plus qu'un client et un serveur à la main (le nombre de canaux requis augmentant très rapidement). Il reste envisageable d'écrire un compilateur qui générerait automatiquement ces canaux frais. Il n'empêche que la complexité induite par les multiples copies des processus et canaux tournant en parallèle nuit aux performances de l'outil et, lorsque celui-ci termine, rend les traces d'attaques très compliquées à lire. Deuxièmement, nous avons identifié plusieurs cas de non-terminaison liés à des attaquants « personnalisés ». En particulier, les attaquants forgeant et rejouant des messages ont la particularité de ne pas être activés par la réception d'un message (ils envoient des messages quand ils veulent). Il s'avère que l'analyse ne termine que lorsqu'ils sont activés à la réception d'un message (de façon similaire à un attaquant modifiant le contenu d'une commande). Cette condition très restrictive rend les actions de forger et rejouer caduques.

Nous avons également réalisé plusieurs essais avec l'outil Tamarin [13][Schmidt et al., 2012, Meier et al., 2013]. Développé durant les thèses de S. Meier et B. Schmidt, cet outil est maintenant maintenu en collaboration entre l'ETH Zurich, l'Université d'Oxford et le Loria Nancy. Comme ProVerif, il est capable de vérifier des protocoles avec un nombre

12. L'oracle transfert les message à l'intrus Dolev-Yao.
13. http://www.infsec.ethz.ch/research/software/tamarin.html

infini de sessions. Les protocoles sont spécifiés sous forme de règles de réécriture de *multisets* et les propriétés sous forme de propriétés de logique temporelle du premier ordre. Cependant nos études menées en collaboration avec les développeurs de l'outil sont pour le moment inconcluantes. En effet, il apparaît que l'algorithme d'analyse de Tamarin ne termine pas sur les modélisations de l'approche A²SPICS. La cause la plus probable serait le fait que les automates représentant les comportements sont cycliques, faisant boucler l'algorithme de recherche arrière de l'outil. Cette hypothèse reste à vérifier.

Enfin, comme détaillé en section 5.6.1, Rocchetto et Tippenhauer parviennent avec succès à vérifier des propriétés de sûreté de fonctionnement à l'aide d'outils de vérification de protocoles cryptographiques. Il sont en mesure d'ajouter des théories équationnelles à l'intrus Dolev-Yao pour le renforcer mais ne parviennent pas à restreindre ses capacités. Il serait donc intéressant de voir s'il est possible d'appliquer les méthodes proposées par le projet AVANTSSAR pour modéliser des protocoles à état dans Tamarin où il semble possible de dégrader l'intrus Dolev-Yao afin d'avoir une modélisation fine des attaquants.

5.6. Conclusion

Dans ce chapitre nous avons présenté l'approche A²SPICS, une approche permettant de vérifier des propriétés de sûreté de fonctionnement en présence d'attaquants. Afin de prendre en compte à la fois sécurité et sûreté, cette approche fonctionne en deux phases, chacune reposant sur une analyse de risques. Une première phase définit des modèles d'attaquants à partir d'une analyse de risques en sécurité. A ce titre, nous avons proposé une méthode permettant d'étudier les attaquants possibles sur un système en termes de position et de capacités [Puys et al., 2016b]. Dans une seconde phase, ces modèles d'attaquants sont utilisés conjointement avec un modèle du comportement du système dans une analyse afin de vérifier si des propriétés de sûreté de fonctionnement peuvent être violées par les attaquants. Dans le cadre du master d'Abdelaziz Khaled, nous avons proposé une première implémentation de cette analyse via le model-checker UPPAAL [Puys et al., 2017]. Nous avons également étudié la possibilité de remplacer UPPAAL par des outils de vérification de protocoles cryptographiques tels que ProVerif et Tamarin.

Plus généralement, l'approche A²SPICS apporte des réponses aux questions suivantes :

— Déterminer si un attaquant, une fois entré dans le système, peut ou non violer des propriétés de sûreté de fonctionnement afin d'impacter le procédé potentiellement critique.

— Déterminer quelles capacités sont nécessaires aux attaquants pour réaliser leurs attaques. Cela permet, en testant différents attaquants à différentes positions et avec différentes capacités, de déterminer quel est le profil minimal de l'attaquant. Par exemple, est-il requis que l'attaquant soit capable de réaliser des opérations cryptographiques ou non ? C'est particulièrement intéressant afin de savoir quelles machines devraient être corrompues par l'attaquant ou à quels endroits du réseau

il devrait se connecter. Ainsi, il est possible de n'apposer que les contre-mesures nécessaires afin d'impacter au minimum le fonctionnement du système.

— Dans une certaine mesure, cette approche permet aussi de comparer les attaquants (et leurs capacités) entre eux. Par exemple, un attaquant ne pouvant faire que du rejeu de message est-il équivalent à un attaquant pouvant modifier des messages ? Bien entendu, ces comparaisons ne sont valides que pour le système étudié et non en général.

En particulier dans le cas de l'attaque Maroochy Shire, le manque de préparation des employés aurait été limité par des analyses telles que celles de l'approche A^{2}SPICS. En effet, des scénarios d'attaques empêchant les pompes de fonctionner auraient pu être découverts par l'analyse et le système aurait pu être protégé. A minima, les employés auraient été plus vigilants lors de l'apparition de défauts au niveau des pompes. Pour conclure, nous comparons l'approche A^{2}SPICS à l'état de l'art en section 5.6.1 puis nous discutons respectivement des résultats et des perspectives en sections 5.6.2 et 5.6.3.

5.6.1. Position de l'approche A^{2}SPICS par rapport à l'état de l'art

Concernant les méthodes d'analyse de risques en sécurité, elles sont essentiellement focalisées avant tout sur les biens à protéger, généralement appelés *assets*. La méthode proposée en section 5.3, quand à elle, se focalise quasi uniquement sur les attaquants. Il est ainsi difficile de la comparer avec les méthodes listées en section 5.1.5.1 car elles n'ont pas les mêmes objectifs. Les méthodes d'analyse de risques connues visent à exhiber des risques, c'est-à-dire l'exploitation de vulnérabilités par des sources de menaces visant des biens à protéger. A l'inverse, notre méthode part de la topologie du système et essaie de définir quelles sont les capacités des attaquants en fonction de leurs positions possibles et s'ils peuvent atteindre des objectifs avec ces capacités. Elle est en ce sens beaucoup plus proche des arbres d'attaques, de menaces, mais se distingue au passage de ces derniers en étant qualitative et non quantitative. A ce titre, l'approche A^{2}SPICS ne donne donc aucune information sur la plausibilité des attaquants mais évite à l'utilisateur de fournir une distribution de probabilité qu'il n'est pas toujours aisé d'obtenir. Il pourrait notamment être intéressant de combiner notre approche avec les approches orientées sur les biens à protéger.

Concernant l'approche A^{2}SPICS en elle-même, elle se distingue de plusieurs approches listées en section 5.1.5.2 qui semblent plutôt ressembler à des approches d'analyse de risques orientées sur les systèmes industriels. C'est principalement le cas de Byres *et al.* [Byres et al., 2004] qui quantifient les critères comme la vraisemblance ou la gravité sur une échelle de un à quatre à la manière d'EBIOS. Par ailleurs, 18 approches parmi les 23 listées par Cherdantseva *et al.* [Cherdantseva et al., 2015] sont quantitatives et nécessitent une distribution de probabilités. Cependant, la quasi totalité des approches probabilistes ne donnent aucune information sur les sources de ces distributions et sur leur fiabilité. De plus il n'est souvent pas aisé de mesurer l'impact des variations de ces probabilités sur les résultats de l'analyse (une légère variation peut-elle modifier sensiblement les résultats). De façon analogue aux méthodes d'analyse de risques, elles ont par contre l'avantage de pouvoir quantifier les résultats. Dans [Kriaa et al., 2015a], Kriaa *et al.* définissent quatre critères pour classifier les approches combinant sécurité et sûreté, à savoir :

1. le fait que l'approche repose sur des modèles formels ;
2. le fait que l'approche soit à la fois qualitative et quantitative ;
3. le fait que l'approche soit automatisée ;
4. le fait que les modèles soient facilement adaptables.

Aucun des travaux référencés par Kriaa *et al.* ne valide par exemple le critère d'être automatisé. L'approche A²SPICS respecte les critères 1, 3 et 4 (reposant sur un outil de vérification formelle et automatisée, UPPAAL et permettant de changer simplement les positions des attaquants et les comportements). On remarquera par ailleurs que l'approche A²SPICS est proche des travaux de Rocchetto et Tippenhauer [Rocchetto and Tippenhauer, 2017] qui semblent également valider les critères 1, 3 et 4. L'approche A²SPICS présentent néanmoins des différences significatives avec cette dernière. Les auteurs utilisent un outil de vérification de protocoles cryptographiques (CL-Atse) et considèrent des variation de l'intrus Dolev-Yao auquel ils ajoutent des théories équationnelles (permettant notamment des interactions physiques avec le procédé [Rocchetto and Tippenhauer, 2016]). A²SPICS s'intéresse plutôt à des attaquants modélisant des sous-ensembles de Dolev-Yao afin d'obtenir une granularité plus fine des capacités d'attaques requises. Par ailleurs à notre connaissance, Rocchetto et Tippenhauer ne prennent pas en compte la topologie du réseau dans leurs analyses.

5.6.2. Remarques sur les résultats

Plusieurs remarques peuvent être tirées des implémentations et des résultats des analyses. Parmi les questions évoquées au début de cette section et auxquelles ce chapitre tente de répondre, il en est une encore largement ouverte : est-il plus simple d'utiliser les outils de vérification de protocoles cryptographiques ou bien les outils de vérification génériques ? Une piste de réponse pourrait être de savoir si les abstractions faites par les outils de vérification de protocoles cryptographiques sont compatibles avec l'approche A²SPICS ? Plusieurs éléments de réponse peuvent être apportés.

Premièrement, les propriétés de sûreté de fonctionnement ciblées nécessitent que les outils permettent de spécifier des propriétés sous forme de prédicats logiques. La plupart des outils ne le permettent pas (ou pas encore) mais on constate globalement que les outils récents comme Tamarin, ou maintenus sur le long terme comme ProVerif, le permettent. On peut donc supposer que le développement des outils de vérification de protocoles cryptographiques vise des objectifs similaires à ceux de l'approche A²SPICS. On peut aussi se demander si l'impossibilité d'utiliser les outils de vérification de protocoles cryptographiques montrée en section 5.5 vient de limites liées aux techniques utilisées par ces derniers ou si c'est le problème abordé par A²SPICS qui est intrinsèquement dur et/ou qui présente des différences notables avec le domaine de la vérification de protocoles cryptographiques.

Enfin, on peut faire quelques remarques un peu plus attendues sur les résultats et les temps de calculs des outils, en particulier dans les quelques cas où les outils de vérification de protocoles cryptographiques ont pu conclure. En effet, on remarque que certains outils ne terminent pas dans certaines configurations alors que d'autres oui, cela change en fonction des attaquants, des topologies, des comportements, des propriétés, etc. C'est une remarque classique faites dans les travaux de benchmark, en fonction

des techniques de vérification utilisées par les outils, leurs conclusions et leur limites peuvent varier. Une autre remarque assez attendue est qu'il semble simple d'exprimer les comportements en UPPAAL mais compliqué d'y décrire des attaquants. C'est assez logique puisque l'outil est destiné à décrire des systèmes et de les vérifier contre des propriétés de sûreté de fonctionnement. La remarque inverse s'applique par ailleurs aux outils de vérification de protocoles cryptographiques.

5.6.3. Limites et travaux futurs

Comme pointé en introduction, le domaine de la sécurité des systèmes industriels est encore jeune et les outils existants manquent de maturité. Dans le cas de l'approche A^{2}SPICS, plusieurs limites restent à dépasser et de nombreuses améliorations sont possibles. Premièrement, l'implémentation avec UPPAAL reste à l'état de preuve de concept et demande encore beaucoup d'ajustements pour être utilisable. En particulier les attaquants restent limités et peuvent aisément boucler s'ils ne sont pas modélisés avec précaution. Nous avons notamment dû borner le nombre maximal d'actions qu'ils peuvent effectuer dans une étape d'une attaque, cette borne étant configurable.

De plus une limite dans la modélisation a pour conséquence de générer des faux négatifs avec l'implémentation en UPPAAL. En effet, dans les résultats de l'étude avec UPPAAL, nous avons pointé que la propriété Φ_1 ne pouvait jamais être violée. Cependant, nous avons également exhibé une attaque contre cette propriété en annexe A.1. Cela vient du fait que deux états du système peuvent être considérés : (i) l'état réel du système (i.e. : si une bouteille est physiquement présente ou non), et (ii) l'état logique (i.e. : si la variable *bottleInPlace* vaut *true* ou *false*). Il apparaît que la modification des capteurs par l'attaquant introduit une décorrélation entre ces deux états (la bouteille serait présente dans l'état logique mais absente en réalité). Cependant, les propriétés sont vérifiées par UPPAAL dans l'état logique (c'est à dire basées sur la valeur des variables), pouvant mener à des attaques manquées (en particulier dans le cas de la propriété Φ_1). L'approche devrait prendre en compte cette décorrélation entre les valeurs des variables et l'état réel du procédé.

Par ailleurs, de plusieurs autres pistes pourraient être explorées. Une première idée serait de renforcer les liens entre la méthode d'analyse de risques montrée en section 5.3 et la seconde phase de l'approche A^{2}SPICS en formalisant clairement les modèles d'attaquants fournis par l'analyse de risques. Il serait aussi intéressant d'étudier la possibilité que des attaquants partagent des connaissances afin de modéliser des collusions. Une autre piste serait de faciliter la modélisation de protocoles de communication. En effet, de nombreux industriels utilisent des protocoles propriétaires et non documentés au lieu de MODBUS et OPC-UA. Enfin, dans la même veine que le chapitre 4, il serait très intéressant d'étudier l'encapsulation de protocoles afin de modéliser des messages MODBUS transportés par OPC-UA ou encore l'extension MODBUS/CAN qui ajoute une fonction à MODBUS afin de transporter des paquets CAN bus.

Troisième partie

Conclusion

Chapitre 6

Conclusion

> There are two ways to write error-free programs ; only the third one works.
>
> *(Alan J. Perlis, 1982)*

Résumé du chapitre

Ce chapitre conclut ce manuscrit en résumant les contributions de la thèse, ainsi que leurs perspectives respectives. On montre également les interactions possibles entre les contributions en décrivant des perspectives d'utilisation conjointe. Enfin, on discute de perspectives générales sur la sécurité des systèmes industriels.

Sommaire

6.1. Résumé des contributions

Cette section résume les contributions de cette thèse et les replace dans l'exemple de l'attaque Maroochy Shire, présentée en section 1.4. Lors de cette attaque, un ancien employé de Hunter Watertech (maintenant HWT), sous-traitant du conseil du Comté de Maroochy Shire (maintenant Sunshine Coast) déverse le contenu d'une cuve d'eaux usagées dans la nature. Durant l'attaque qui s'est déroulée sur plusieurs mois, le système d'épuration ciblé a connu une série de défauts, en particulier :

— des pompes ne fonctionnant pas lorsqu'elles auraient dû ;

— des alarmes n'étant pas remontées au SCADA ;

— le SCADA n'étant pas en mesure de communiquer avec des pompes.

A la suite de ces défauts (aujourd'hui connus comme étant les conséquences de l'attaque), une cuve a débordé. A notre connaissance, aucune analyse de l'attaque [Slay and Miller, 2007, Abrams and Weiss, 2008, Weiss, 2010] ne décrit précisément quel enchaînement d'actions a conduit au débordement de la cuve. Cependant, au vu des défauts qu'a connu le système, on peut *supposer* qu'une pompe était censée vider la cuve à mesure qu'elle se remplissait d'eaux usagées. Le fait qu'elle ne démarre pas aurait fait déborder la cuve. Par ailleurs, dû au fait que les alarmes n'étaient pas remontées au SCADA, les opérateurs ne se seraient aperçus des problèmes qu'une fois les dégâts causés.

6.1.1. Le projet ARAMIS : un dispositif de filtrage

Lors de l'attaque Maroochy Shire, il parait évident que rien n'a empêché les pompes de s'arrêter alors qu'elles auraient dû fonctionner. Une réponse possible à ce problème est donnée en chapitre 3 à travers le projet ARAMIS [ARAMIS, 2014]. L'objectif de ce projet est de proposer un dispositif permettant de cloisonner physiquement les réseaux des systèmes industriels et de filtrer les échanges en tenant compte des contraintes métier. Ce dispositif doit ainsi rejeter tout flux identifié comme interdit, donc potentiellement malveillant. Étant un dispositif embarqué, il doit respecter des contraintes de mémoire en plus des contraintes de temps des communications industrielles décrites en section 1.1.1. Parmi l'ensemble des composants du dispositif, nous nous sommes focalisés sur le filtre et ses langages d'entrée. Les règles garanties par le prototype de filtre présenté permettent notamment d'expliciter les requêtes que les clients peuvent effectuer sur chaque variable de chaque serveur. Elles prennent aussi en compte des propriétés orientées métier sur les valeurs possibles des variables, sur la temporisation des commandes, et sur l'état global du système [Puys et al., 2016c, Badrignans et al., 2017]. Nous avons présenté les deux langages d'entrée du filtre, à savoir un langage dit « bas niveau » et une API Python. Nous également proposé des fonctionnalités de l'API permettant de générer des règles en langage bas niveau. De façon plus générale, nous avons montré dans ce chapitre comment un filtre applicatif et (ici le filtre du dispositif de filtrage) peut aider à endiguer une variante de l'attaque Maroochy Shire en filtrant les commandes ayant arrêté la pompe.

6.1.2. Vérification formelle de protocoles de communication industriels

Les automates programmables utilisés durant l'attaque Maroochy Shire communiquaient via les protocoles de communication industriels MODBUS et DNP3. Ces deux protocoles, comme beaucoup d'autres du domaine, ne fournissent aucune protection contre des attaquants. C'est un problème car un attaquant capable de communiquer avec les automates peut donc lancer arbitrairement des commandes affectant le procédé. En particulier, il pourrait se faire passer pour le client et lancer une commande afin d'arrêter la pompe lorsque la cuve est pleine. Pour pallier ce risque, plusieurs travaux académiques ont proposé d'ajouter de la sécurité à MODBUS. Par ailleurs, le protocole OPC-UA a notamment été développé à partir de son prédécesseur OPC et implémente des mesures de sécurité prétendues à l'état de l'art des attaques. Cependant, comment s'assurer que ces protocoles garantissent effectivement des propriétés de sécurité ? Une possibilité consiste à décrire les protocoles dans certains formalismes permettant de vérifier si un attaquant peut mettre en défaut les propriétés de sécurité revendiquées. Ce domaine est nommé vérification de protocoles cryptographiques et beaucoup d'outils ont été développés pour automatiser ces analyses.

Si la vérification de protocoles cryptographiques s'est largement développée pour des protocoles « grand public » comme SSH ou TLS, elle peine encore à s'imposer dans le cas des protocoles de communication industriels. Il nous parait cependant très dommageable que des protocoles contrôlant des systèmes aussi critiques que des centrales nucléaires ou des barrages puissent n'avoir jamais été vérifiés formellement. Pour cela le chapitre 4 s'est focalisé sur l'application des techniques de vérification de protocoles cryptographiques aux protocoles de communication industriels. Dans un premier temps, nous avons réalisé une analyse de propriétés de secret et d'authentification sur les protocoles de handshake d'OPC-UA à l'aide de l'outil ProVerif en modélisant ces protocoles à partir du standard. Ces travaux ont donné lieu à une publication à la conférence SAFECOMP 2016 [Puys et al., 2016a]. Dans un second temps, nous avons introduit une formalisation de propriétés basées sur l'ordre des messages, actuellement inexistantes dans les outils de vérification. Ces travaux ont donné lieu à une publication à la conférence SECRYPT 2017 [Dreier et al., 2017b].

6.1.3. Recherche automatique de scénarios d'attaques applicatives

Lors de l'attaque Maroochy Shire, les employés n'étaient pas préparés à l'attaque qui est survenue. On peut aisément s'en rendre compte du fait que non seulement l'attaque s'est déroulée sur plusieurs mois et que les conséquences étaient visibles dès les premières actions. Afin de préparer au mieux les opérateurs, il convient au préalable d'identifier les faiblesses du système. Pour cela, plusieurs méthodes d'analyse de risques ont été proposées au fil des années. Certaines comme EBIOS [ANSSI, 2010] et MEHARI [CLUSIF, 2010] en France, visent à identifier les risques en matière de sécurité pesant sur les systèmes d'information. A l'opposé, d'autres méthodes comme HAZOP [Kletz, 1999, IEC-61882, 2001] et AMDEC [IEC-60812, 1985] évaluent les risques en matière de sûreté. A notre connaissance, ces approches sont souvent très spécifiques à la sécurité ou bien à la sûreté. Elles sont aussi généralement très structurées mais souvent manuelles. Aussi, plusieurs outils ont été développés pour aider à les appliquer.

Dans le chapitre 5, nous proposons l'approche A^{2}SPICS, pour *Applicative Attack Scenarios Production for Industrial Control Systems*. Cette méthode d'analyse consiste à utiliser des techniques de vérification par simulation telles que le model-checking afin de vérifier si un système assure des propriétés de sûreté en présence d'attaquants. Afin de prendre en compte à la fois sécurité et sûreté, A^{2}SPICS fonctionne en deux phases, chacune reposant sur une analyse de risques. Dans une première phase, on s'intéresse à une analyse de risques en terme de sécurité basée sur la topologie du réseau, sur les objectifs des attaquants potentiels, ainsi que sur les fonctionnalités de sécurité apportées par les protocoles de communication. A l'issue de cette analyse, des *modèles d'attaquants* sont produits. Ces modèles détaillent notamment les positions, capacités et objectifs des attaquants. Cette première partie de l'approche a fait l'objet d'un article à la conférence française AFADL 2016 [Puys et al., 2016b].

Dans une seconde phase, on tire avantage du fait que les systèmes industriels sont très bien analysés en terme de sûreté de fonctionnement. Aidés par les propriétés à garantir, qui résultent de ces analyses de sûreté, on s'intéresse à voir si les modèles d'attaquants produits par la phase 1 sont en mesure de violer des propriétés de sûreté de fonctionnement. On prend pour cela en compte le comportement du procédé ainsi que les valeurs des variables manipulées par les serveurs. Par exemple dans le cas d'un haut fourneau, une propriété à vérifier est que le four ne démarre pas si la porte est ouverte. Via une méthode de vérification, on peut alors vérifier, pour chaque propriété, si un attaquant peut la violer. Dans le cadre du master d'Abdelaziz Khaled, une première implémentation de cette analyse a été proposée via le model-checker UPPAAL. Nous avons également étudié la possibilité de remplacer UPPAAL par des outils de vérification de protocoles cryptographiques tels que ProVerif et Tamarin. Au regard de l'attaque Maroochy Shire, l'approche A^{2}SPICS apporte des réponses aux questions suivantes :

— Déterminer si un attaquant, une fois entré dans le système, peut ou non violer des propriétés de sûreté de fonctionnement afin d'impacter le procédé. Autrement dit, dans le cas de l'attaque Maroochy Shire, est-il en mesure de faire déborder la cuve en empêchant les pompes de fonctionner ?

— Déterminer quelles capacités sont nécessaires aux attaquants pour réaliser leurs attaques. Ainsi, il est possible de n'apposer que les contre-mesures nécessaires afin d'impacter au minimum le fonctionnement du système.

6.2. Perspectives d'utilisation conjointe des contributions

Nous avons décrit des perspectives pour chaque contribution dans leurs chapitres respectifs. Une question intéressante peut être de regarder comment elles peuvent s'articuler entre elles.

Vérification du dispositif de filtrage à l'aide de l'approche A^{2}SPICS : Une première idée serait d'utiliser l'approche A^{2}SPICS pour vérifier le dispositif de filtrage. En le décrivant comme un des processus du système, avec à la fois l'algorithme de filtrage et les règles qui y sont configurées, on peut ainsi tester les propriétés de sûreté avec et sans le dispositif. Si celui-ci fonctionne comme attendu et qu'il est bien configuré, les attaques potentiellement trouvées sur le système disparaîtront lorsque le dispositif

est inclus. L'approche A^{2}SPICS peut aussi servir de simulateur afin d'aider à entrer les règles nécessaires pour éviter les attaques, mais aussi de vérifier si des règles ne sont pas superflues afin de limiter la latence introduite par le filtre.

Utilisation des outils de vérification de protocoles pour l'approche A^{2}SPICS : L'approche A^{2}SPICS se base sur les fonctionnalités de sécurité apportées par les protocoles pour déterminer quelles sont les capacités des attaquants. Cependant, ces hypothèses sur les attaquants seront caduques si les propriétés de sécurité des protocoles ne sont pas vérifiées formellement. La question serait donc comment interpréter formellement les résultats des analyses de protocoles afin d'en déduire les capacités des attaquants ? Cela requiert une analyse fine de l'expression des propriétés de sécurité à tester pour les protocoles (savoir précisément quel vecteur d'attaque découle de l'absence de chaque propriété).

Analyses conjointes des systèmes en terme de sûreté et de sécurité : L'idée sous-jacente au fait d'utiliser des outils de vérification de protocoles cryptographiques dans le cadre de l'approche A^{2}SPICS ne se limite pas au fait de tirer parti du fait que les outils implémentent la logique de l'intrus dans les analyses. Il serait particulièrement intéressant que les analyses réalisées puissent impliquer à la fois des propriétés de sécurité et de sûreté. Plus précisément, les attaquants étant modélisés en fonction des résultats des vérifications de protocoles cryptographiques, on doit déduire leurs capacités des résultats des analyses comme pointé dans le paragraphe précédent. Une autre possibilité serait de modéliser à la fois le système industriel comme dans l'approche A^{2}SPICS, et les protocoles de communication dans le même modèle. On aurait de cette façon l'attaquant qui exploite réellement des failles de sécurité des protocoles pour attaquer des propriétés de sûreté sans les indirections induites par le fait que l'approche est en deux phases (en particulier l'abstraction des vecteurs d'attaques évoquée ci-dessus).

6.3. Perspectives du domaine

Il va aujourd'hui sans dire que la sécurité des systèmes industriels est un enjeu crucial de notre société. La liste des attaques que nous avons présentée en section 1.4 est loin d'être exhaustive et bien d'autres viendront certainement s'y ajouter dans le futur. Ces trois axes de travail sur le sujet nous auront néanmoins permis de remarquer la facilité avec laquelle le grand public se laisse convaincre de l'importance de ce domaine de recherche. Il parait en effet évident pour tout le monde que le fait qu'un « pirate » puisse plonger un pays dans le noir n'est pas quelque chose d'acceptable et que des moyens doivent être alloués à la protection des sites sensibles. C'est une conclusion qui a bien été remarquée par de nombreux gouvernements. La protection des infrastructures critiques est donc devenue une mission d'importance pour les agences gouvernementales. En particulier en France, l'ANSSI en a fait l'un de ses chevaux de bataille et communique activement sur le sujet avec des guides de bonnes pratiques, des séminaires et des interventions dans les médias.

Il reste néanmoins que ce domaine est très jeune et manque de maturité. Si les contributions du chapitre 3 visent directement à être appliquées dans l'industrie, les autres

contributions restent encore à être appliquées dans le monde réel. Cela demandera d'une part le temps que les industriels soient convaincus de l'importance de ces recherches et d'autre part que les scientifiques aient fait des efforts suffisants de vulgarisation et d'expérimentation pour les rendre applicables. La route est encore longue pour transférer dans l'industrie tout le savoir accumulé par les scientifiques mais le jeu n'en vaut-il pas la chandelle ?

Quatrième partie

Annexes

Annexe A

Exemples de scénarios d'attaques contre des systèmes industriels

Cet appendice a pour but de présenter des exemples de scénarios d'attaques simples sur des systèmes industriels qui seront utilisés tout au long du manuscrit. On considère des attaques au niveau du réseau où l'attaquant forge, modifie ou rejoue des paquets afin d'impacter la valeur des variables sur le serveur, et de modifier le procédé. On considère également que lorsqu'un client est corrompu (contrôlé par l'attaquant), celui-ci est en mesure d'envoyer les commandes de son choix. Lorsqu'un serveur est corrompu, l'attaquant a le plein contrôle sur les variables contrôlées par celui-ci ainsi que sur les requêtes arrivantes et les réponses sortantes. Il n'est cependant pas en mesure de forger de nouveaux messages. Pour chaque exemple, on présentera :

— une description du procédé ;
— une ou plusieurs propriétés de sûreté à garantir ;
— une liste non exhaustive d'attaques informatiques pouvant violer ces propriétés ;
— différentes variations de topologies réseau pouvant représenter le système.

Ensuite pour chaque topologie, on listera :

— les protocoles de communication utilisés ;
— la ou les positions de l'attaquant et ces capacités ;
— les attaques existantes s'il y en a.

A.1. Chaîne de mise en bouteille

Cet exemple provient du simulateur de procédé VirtualPlant[1] et est présenté en figure A.1. Des bouteilles avancent sur un tapis roulant. Un capteur détecte lorsqu'elles sont en position, arrête le tapis roulant et actionne une vanne faisant couler le liquide. Un autre capteur permet de détecter lorsque la bouteille est pleine. La vanne se referme, et le tapis roulant redémarre.

Plusieurs propriétés peuvent être envisagées telles que le fait que la vanne ne s'ouvre que lorsqu'une bouteille est en position et que le tapis roulant est à l'arrêt. Une propriété complémentaire pourrait être que les bouteilles ayant dépassées la vanne soient bien remplies.

1. https://github.com/jseidl/virtuaplant

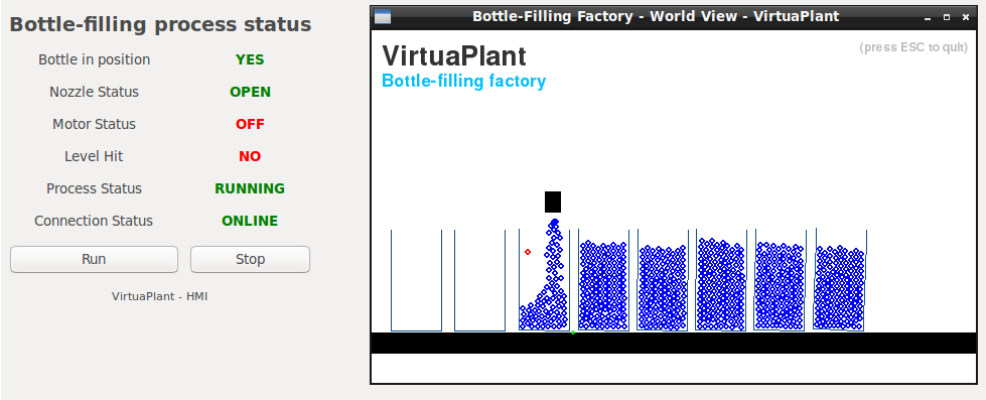


FIGURE A.1. – Chaîne de mise en bouteille

Attaque 1 : Forcer l'ouverture de la vanne alors qu'aucune bouteille n'est présente en dessous.

Attaque 2 : Forcer le tapis roulant à avancer quel que soit l'état des capteurs. Les bouteilles avanceront vides.

Attaque 3 : Forcer l'ouverture de la vanne alors que le tapis roulant est en mouvement. Le liquide se répandra hors des bouteilles.

Attaque 4 : Forcer l'ensemble du procédé à l'arrêt. Cette attaque ne viole aucune des propriétés de sûreté énoncées ci-dessus mais impacte la disponibilité du procédé.

A.1.1. Variables en lecture/écriture

Le moteur, la vanne et les capteurs sont contrôlés par des variables booléennes. Une cinquième variable également booléenne démarre ou arrête l'ensemble du procédé. Dans le simulateur VirtualPlant, toutes ces variables (capteurs comme actionneurs) sont accessibles en écriture afin de faciliter la démonstration des attaques. Dans toutes les topologies de cet exemple, si l'attaquant est le client corrompu, toutes les attaques sont toujours possibles, celui-ci pouvant manipuler toutes les variables à sa guise.

Topologie 1 : Un seul serveur MODBUS ou OPC-UA (configuré en mode None) contrôle l'ensemble du procédé. Il communique avec un client MODBUS ou OPC-UA qui contrôle l'arrêt et le démarrage du procédé et lit l'état des capteurs, du tapis roulant et de la vanne. Quelque soit la position de l'attaquant, toutes les attaques sont possibles.

Topologie 2 : Un seul serveur OPC-UA (configuré en mode Sign ou SignEncrypt) contrôle l'ensemble du procédé et communique avec un client OPC-UA qui contrôle l'arrêt et le démarrage du procédé et lit l'état des capteurs, du tapis roulant et de la vanne. Si l'attaquant est le serveur corrompu, toutes les attaques sont possibles. Dans le cas où l'attaquant est uniquement en MITM, aucune attaque n'est possible.

Topologie 3 : Le tapis roulant et son capteur sont contrôlés par un serveur MODBUS ou OPC-UA (configuré en mode None). La vanne et son capteur par un serveur OPC-UA (configuré en mode Sign ou SignEncrypt). Un unique client multi-protocoles communique avec les deux serveurs. Si l'attaquant est le serveur OPC-UA corrompu, toutes les attaques sont possibles :

— Attaque 1 : l'attaquant contrôle la vanne.

— Attaque 2 : l'attaquant contrôle le capteur de niveau et peut faire croire au tapis roulant que les bouteilles sont toujours pleines.

— Attaque 3 : l'attaquant contrôle la vanne.

— Attaque 4 : l'attaquant contrôle le capteur de niveau et peut faire croire au tapis roulant que les bouteilles sont toujours vide, bloquant le tapis roulant qui attendra qu'elles se remplissent.

Réciproquement, si l'attaquant est le serveur MODBUS ou en MITM, toutes les attaques sont également possibles :

— Attaque 1 : l'attaquant contrôle le capteur de position et peut faire croire à la vanne que les bouteilles sont toujours en position, la vanne s'ouvrira pour laisser couler le liquide.

— Attaque 2 : l'attaquant contrôle le tapis roulant.

— Attaque 3 : l'attaquant contrôle le tapis roulant.

— Attaque 4 : l'attaquant contrôle le capteur de position et peut faire croire à la vanne que les bouteilles ne sont jamais en position, bloquant la vanne qui attendra qu'elles soient bien placées.

Topologie 4 : Le tapis roulant et son capteur sont contrôlés par un serveur OPC-UA (configuré en mode Sign ou SignEncrypt). La vanne et son capteur par un serveur MODBUS ou OPC-UA (configuré en mode None). Un unique client multi-protocoles communique avec les deux serveurs. L'analyse de la topologie 3 s'applique de façon analogue.

A.1.2. Variables en lecture seule

Afin de rendre cette analyse plus plausible, nous considérerons maintenant les variables du moteur, de la vanne et les capteurs en lecture seule dû au fait qu'elles ne sont pas sensées être contrôlées par un client. Dans toutes les topologies de cet exemple, si l'attaquant est le client corrompu, alors seule l'attaque 4 est possible car la variable contrôlant le démarrage où l'arrêt du procédé est la seule accessible en écriture.

Topologie 1 : Un seul serveur MODBUS ou OPC-UA (configuré en mode None) contrôle l'ensemble du procédé. Il communique avec un client MODBUS ou OPC-UA qui contrôle l'arrêt et le démarrage du procédé et lit l'état des capteurs, du tapis roulant et de la vanne. Si l'attaquant est le serveur corrompu, toutes les attaques sont possibles. Dans le cas où l'attaquant est uniquement en MITM, seule l'attaque 4 est possible.

Topologie 2 : Un seul serveur OPC-UA (configuré en mode Sign ou SignEncrypt) contrôle l'ensemble du procédé et communique avec un client OPC-UA qui contrôle l'arrêt et le démarrage du procédé et lit l'état des capteurs, du tapis roulant et de la vanne. Si l'attaquant est le serveur corrompu, toutes les attaques sont possibles. Dans le cas où l'attaquant est uniquement en MITM, aucune attaque n'est possible.

Topologie 3 : Le tapis roulant et son capteur sont contrôlés par un serveur MODBUS ou OPC-UA (configuré en mode None) et la vanne et son capteur par un serveur OPC-UA (configuré en mode Sign ou SignEncrypt). Un unique client multi-protocoles communique avec les deux serveurs. Si l'attaquant est le serveur OPC-UA corrompu, toutes les attaques sont possibles :

— Attaque 1 : l'attaquant contrôle la vanne.

— Attaque 2 : l'attaquant contrôle le capteur de niveau et peut faire croire au tapis roulant que les bouteilles sont toujours pleines.

— Attaque 3 : l'attaquant contrôle la vanne.

— Attaque 4 : l'attaquant contrôle le capteur de niveau et peut faire croire au tapis roulant que les bouteilles sont toujours vide, bloquant le tapis roulant qui attendra qu'elles se remplissent.

Réciproquement, si l'attaquant est le serveur MODBUS, toutes les attaques sont également possibles :

— Attaque 1 : l'attaquant contrôle le capteur de position et peut faire croire à la vanne que les bouteilles sont toujours en position, la vanne s'ouvrira pour laisser couler le liquide.

— Attaque 2 : l'attaquant contrôle le tapis roulant.

— Attaque 3 : l'attaquant contrôle le tapis roulant.

— Attaque 4 : l'attaquant contrôle le capteur de position et peut faire croire à la vanne que les bouteilles ne sont jamais en position, bloquant la vanne qui attendra qu'elles soient bien placées.

Dans le cas où l'attaquant est uniquement en MITM, soit l'attaque 4 est possible, soit aucune attaque n'est possible suivant sur quel serveur se trouve la variable contrôlant le démarrage et l'arrêt du procédé.

Topologie 4 : Le tapis roulant et son capteur sont contrôlés par un serveur OPC-UA (configuré en mode Sign ou SignEncrypt) et la vanne et son capteur par un serveur MODBUS ou OPC-UA (configuré en mode None). Un unique client multi-protocoles communique avec les deux serveurs. L'analyse de la topologie 3 s'applique en miroir.

On remarque que dans cette configuration, l'analyse est similaire mais requiert pour les attaques 1, 2 et 3 qu'un serveur soit corrompu. Cette contrainte importante pour l'attaquant motive le fait que des variables ne devant pas être écrites soit configurées en lecture seule. Malheureusement, cette pratique est rarement appliquée.

A.2. Sectionneur électrique

Cet exemple que nous avons introduit dans l'article [Puys et al., 2016c] est un cas d'étude du domaine du transfert d'électricité. Un sectionneur S électrique est connecté à un disjoncteur D_1 d'un côté et à deux disjoncteurs D_2 et D_3 de l'autre. Tous les organes (disjoncteurs et sectionneurs) sont représentés par des variables booléennes. Le sectionneur est un organe particulier qui ne doit pas recevoir de commande d'ouverture lorsque du courant le traverse. Cela signifie que soit D_1, soit D_2 et D_3 doivent être ouverts au préalable. Dans le cas contraire un arc électrique peut apparaître et causer des dégâts aux humains et aux matériels environnants. Par ailleurs tous les disjoncteurs et le sectionneur ne doivent pas recevoir de commande d'ouverture (resp. de fermeture) lorsqu'ils sont déjà ouverts (resp. fermés). A noter que les liens entre les différents organes de la figure A.2 sont des câbles électriques et non des connexions réseau. Celles-ci dépendent des topologies décrites ci-après.

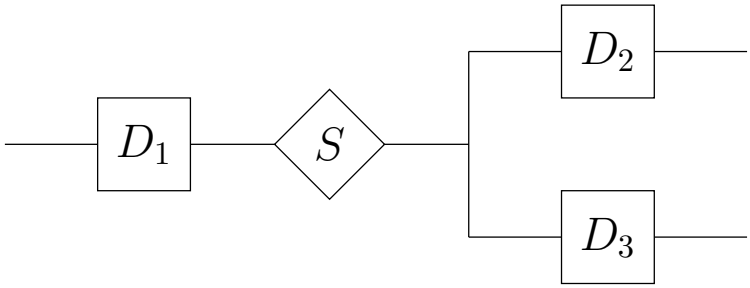


FIGURE A.2. – Sectionneur électrique

Attaque 1 : Forcer l'ouverture de S alors que D_1 et soit D_2 soit D_3 sont fermés.

Attaque 2 : Forcer l'ouverture (resp. la fermeture) d'un organe alors qu'il est déjà ouvert (resp. fermé).

Topologie 1 : Un seul serveur MODBUS contrôle l'ensemble du procédé et communique avec un client MODBUS qui contrôle les organes et lit leur état. L'attaquant est soit le client corrompu, soit le serveur corrompu, soit en MITM. Toutes les attaques sont possibles.

Topologie 2 : Un seul serveur OPC-UA (configuré en mode Sign ou SignEncrypt) contrôle l'ensemble du procédé et communique avec un client OPC-UA qui contrôle les organes et lit leur état. Si l'attaquant est soit le client corrompu soit le serveur corrompu, toutes les attaques sont possibles. Dans le cas où l'attaquant est uniquement en MITM, aucune attaque n'est possible.

Topologie 3 : Tous les disjoncteurs sont contrôlés par des serveurs OPC-UA (configurés en mode Sign ou SignEncrypt) et le sectionneur est contrôlé par un serveur MODBUS. Un unique client multi-protocoles communique avec les deux serveurs. Si

l'attaquant est le client corrompu, toutes les attaques sont possibles. Si l'attaquant est le serveur MODBUS corrompu ou en MITM, l'attaque 2 est évidemment possible puisque l'attaquant a le plein contrôle sur le sectionneur. L'attaque 1 est également possible si l'attaquant attends que les disjoncteurs (qu'il ne peut contrôler) soient dans un état faisant passer le courant au travers du sectionneur. Il peut alors lancer la commande d'ouverture.

Si l'attaquant est le serveur OPC-UA corrompu, l'attaque 2 est évidemment possible puisque l'attaquant a le plein contrôle sur les disjoncteurs. L'attaque 1 est également possible si l'attaquant est capable de retarder des messages MODBUS. Cela suppose par exemple qu'il contrôle un routeur par lequel les message passent et qu'il peut comprendre les messages afin de savoir lesquels retarder. Il a alors la capacité de les retarder ou de les supprimer mais pas de les modifier ou d'en forger de nouveau. L'attaquant peut attendre que S soit ouvert par le client. Il retarde le message d'ouverture le temps de commander les disjoncteurs de façon à ce que le courant traverse S. Si l'attaquant ne peut pas retarder les messages OPC-UA, alors l'attaque 1 n'est pas possible.

Topologie 4 : Le sectionneur, D_2 et D_3 sont contrôlés par un serveur OPC-UA (configuré en mode Sign ou SignEncrypt) et D_1 est contrôlé par un serveur MODBUS. Un unique client multi-protocoles communique avec les deux serveurs. Si l'attaquant est le client corrompu, toutes les attaques sont possibles. Si l'attaquant est le serveur MODBUS corrompu ou en MITM, l'attaque 2 est évidemment possible puisque l'attaquant a le plein contrôle sur D_1. L'attaque 1 est également possible si l'attaquant est capable de retarder des messages OPC-UA. Cela suppose par exemple qu'il contrôle un routeur par lequel les message passent et qu'il peut comprendre les messages afin de savoir lesquels retarder (vraisemblable en mode Sign). Il a alors la capacité de les retarder ou de les supprimer mais pas de les modifier ou d'en forger de nouveau. L'attaquant peut alors attendre que D_1 puis S soient ouvert par le client. Il retarde le message d'ouverture de S le temps de refermer D_1. Si l'attaquant ne peut pas retarder les messages OPC-UA, alors l'attaque 1 n'est pas possible. La même analyse peut se faire réciproquement avec le serveur OPC-UA corrompu.

A.3. Maroochy Shire

L'attaque dite de « Maroochy Shire » est un exemple réel d'attaque visant un système industriel et ayant été perpétrée en 2000. C'est l'exemple introducteur de ce manuscrit qui est décrit en section 1.4. Un ancien employé de Hunter Watertech (maintenant HWT), sous-traitant du conseil du Comté de Maroochy Shire (maintenant Sunshine Coast) déverse le contenu d'une cuve d'eaux usagées dans la nature. Il utilise pour cela du matériel provenant de HWT à savoir un RTU PDS Compact 500 (le même que ceux contrôlant les cuves), un émetteur-récepteur radio (nécessaire aux communications sans-fil) et un ordinateur portable avec un logiciel de HWT servant à reprogrammer les RTU [Weiss, 2010]. Le RTU PDS Compact 500 de HWT communique via les protocoles MODBUS et DNP3 [Clarke et al., 2004]. Durant l'attaque qui s'est déroulée sur plusieurs mois, le système d'épuration ciblé a connu une série de défauts incluant :

— des pompes qui ne fonctionnaient pas lorsqu'elles auraient dû ;

— des alarmes qui n'étaient pas remontées au SCADA ;

— le SCADA qui n'était pas en mesure de communiquer avec des pompes.

A la suite de ces défauts (aujourd'hui connus comme étant les conséquences de l'attaque), une cuve a débordé. D'après les conclusions d'un représentant de l'Agence Australienne de Protection de l'Environnement (EPA), l'eau de la crique est devenue noire, causant plusieurs inondations dans des sous-sols et provoquant la mort de la faune marine. A notre connaissance, aucune analyse de l'attaque [Slay and Miller, 2007, Abrams and Weiss, 2008, Weiss, 2010] ne décrit précisément quel enchaînement d'actions a conduit au débordement d'une cuve. Cependant, au vu des défauts qu'a connu le système, on peut *supposer* qu'une pompe était sensée vider la cuve à mesure qu'elle se remplissait d'eaux usagées. Le fait qu'elle ne démarre pas aurait fait déborder la cuve. Par ailleurs, dû au fait que les alarmes n'étaient pas remontées au SCADA, les opérateurs ne se seraient aperçus des problèmes qu'une fois les dégâts causés. Cette attaque, sous hypothèse de cette suite d'actions, est présentée en figure A.3.

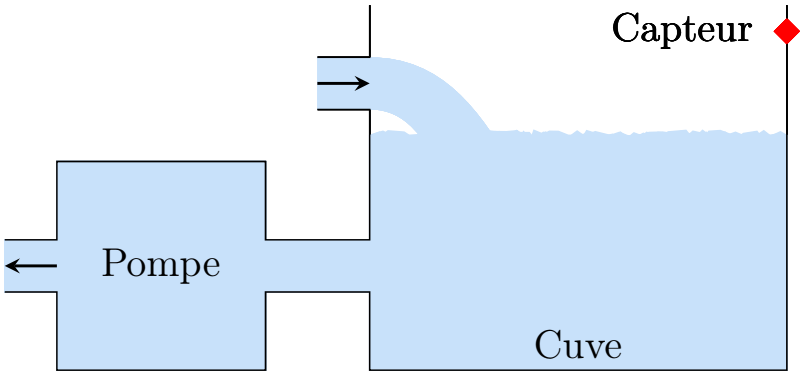


FIGURE A.3. – Attaque Maroochy Shire

Du liquide remplit constamment la cuve que la pompe vide périodiquement. La pompe est contrôlée par une variable booléenne qui décrit si elle fonctionne ou non (il ne parait pas pertinent qu'elle puisse tourner à l'envers comme dans l'exemple A.4 bien que cela ne change pas l'analyse). Un capteur de niveau se trouve en haut de la cuve et est contrôlé par une variable booléenne. On souhaite garantir que si le capteur de niveau en haut de la cuve prend pour valeur True, alors la pompe se met en marche pour faire baisser le niveau de liquide.

Attaque 1 : Empêcher la pompe de démarrer lorsque la cuve est pleine.

Attaque 2 : Faire croire au SCADA que la cuve est toujours vide.

Les analyses de l'attaque [Slay and Miller, 2007, Abrams and Weiss, 2008, Weiss, 2010] peuvent donner lieu à plusieurs interprétations. L'attaquant peut se présenter soit en MITM soit comme un client corrompu (ici un opérateur malveillant) envoyant des commandes malicieuses au serveur, et ce, que la pompe soit mise en marche automatiquement par le serveur ou manuellement par un client légitime. Il se peut également que

la pompe ne puisse être mise en marche qu'automatiquement par le serveur (de façon similaire au tapis roulant et à la vanne de l'exemple A.1). Dans ce cas, l'attaquant a pu reprogrammer le serveur au moyen du logiciel du fabriquant retrouvé sur son ordinateur. On peut alors le considérer comme un serveur corrompu.

Topologie 1 : Un seul serveur MODBUS ou OPC-UA (configuré en mode None) contrôle l'ensemble du procédé. Il communique avec un client MODBUS ou OPC-UA qui contrôle la pompe et lit l'état du capteur. Quel que soit la position de l'attaquant, toutes les attaques sont possibles.

Topologie 2 : Un seul serveur MODBUS ou OPC-UA (configuré en mode None) contrôle l'ensemble du procédé. Il communique avec un client MODBUS ou OPC-UA qui lit l'état de la pompe et du capteur sans pouvoir les contrôler manuellement (on peut imaginer qu'un technicien doive se déplacer sur site pour les maintenances). Si l'attaquant est le serveur corrompu, toutes les attaques sont possibles. Sinon aucune attaque n'est possible.

A.4. Variation de l'attaque Maroochy Shire pour le dispositif de filtrage

Cet exemple présenté en figure A.4 décrit une variante de l'attaque Maroochy Shire utilisée pour décrire le filtre présenté en chapitre 3. Dans cet exemple, la pompe est contrôlée par une variable ternaire ns=5 ;s=Pompe.Etat qui permet de l'arrêter (0) ou bien de la mettre en marche dans un sens ou dans l'autre (1 ou -1). Elle a un temps de latence d'une minute à chaque changement d'état pendant lequel elle ne doit pas recevoir de nouvelle requête. De même que dans l'attaque originale, la cuve est équipée d'un capteur de niveau de liquide, représenté par une variable booléenne en lecture seule ns=5 ;s=Cuve.Niveau.

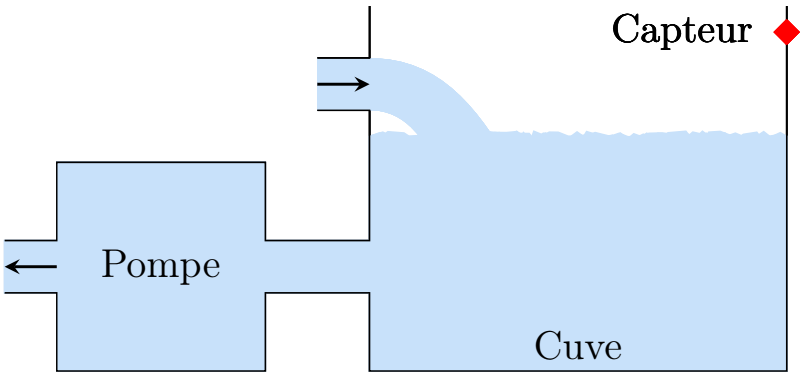


FIGURE A.4. – Variation de l'attaque Maroochy Shire

Les propriétés à garantir pour cet exemple sont les suivantes :

1. Seul le protocole OPC-UA est autorisé en lecture et en écriture.

2. Le(s) client(s) et le(s) serveur(s) contrôlant la pompe ne peuvent parler qu'entre eux,
3. Seule la variable ns=5 ;s=Pompe.Etat contrôlant l'état de la pompe peut être écrite.
4. Les valeurs écrites sur la variable ns=5 ;s=Pompe.Etat ne peuvent être que -1, 0 ou 1.
5. Lorsque la variable ns=5 ;s=Pompe.Etat change de valeur, elle ne doit pas recevoir de nouvelle requête pendant une minute, le temps que la pompe prenne en compte le changement.
6. Le(s) client(s) et le(s) serveur(s) ne doivent pas envoyer plus de 100 requêtes ou réponses par minute.
7. La pompe ne doit pas être arrêtée si la cuve est pleine. Cela signifie que la variable ns=5 ;s=Pompe.Etat ne doit pas être mise à 0 si la variable ns=5 ;s=Cuve.Niveau vaut True.

Attaque 1 : Un autre protocole qu'OPC-UA est utilisé sur le réseau (par exemple MODBUS).

Attaque 2 : Le(s) client(s) contrôlant la pompe envoie(nt) une requête à un autre serveur qu'à celui contrôlant la pompe. Dans une autre version de cette attaque, le(s) serveur(s) contrôlant la pompe envoie(nt) une réponse à d'autres clients que ceux les contrôlant.

Attaque 3 : Écrire dans la variable ns=5 ;s=Cuve.Niveau.

Attaque 4 : Écrire une valeur différente de -1, 0 ou 1 dans la variable ns=5 ;s=Pompe.Etat.

Attaque 5 : Changer l'état de la pompe plus d'une fois en moins d'une minute.

Attaque 6 : Envoyer plus de 100 requêtes ou réponses par minute.

Attaque 7 : Arrêter la pompe si la cuve est pleine, cela signifie forcer la variable ns=5 ;s=Pompe.Etat à 0 alors que la variable ns=5 ;s=Cuve.Niveau vaut True.

Quelque soit la topologie, l'attaque 1 est toujours possible si l'attaquant est un client corrompu ou en MITM, en effet, celui-ci peut par exemple tenter d'initier une connexion SSH avec le serveur, ce qui est interdit par la propriété 1. Si l'attaquant est un serveur corrompu, il peut par exemple répondre au client dans le mauvais protocole en espérant provoquer une erreur. De même l'attaque 2 est toujours possible si l'attaquant est un client ou un serveur corrompu.

Topologie 1 : Un seul serveur MODBUS ou OPC-UA (configuré en mode None) contrôle l'ensemble du procédé. Il communique avec un client MODBUS ou OPC-UA qui contrôle la pompe et lit l'état du capteur. Quel que soit la position de l'attaquant, toutes les attaques sont possibles.

Topologie 2 : Un seul serveur OPC-UA (configuré en mode Sign ou SignEncrypt) contrôle l'ensemble du procédé et communique avec un client OPC-UA qui contrôle la pompe et lit l'état du capteur. Si l'attaquant est le client ou le serveur corrompu, toutes les attaques sont possibles. S'il est en MITM, seule l'attaque 1 est possible.

Annexe B

Fichiers sources listés

Cet appendice a pour but de présenter la version complète des fichiers contenant des codes sources listés tout au long du manuscrit.

B.1. Espace d'adressage OPC-UA pour l'exemple de l'attaque Maroochy Shire

```
<?xml version='1.0' encoding='utf-8'?>
<UANodeSet xmlns="http://opcfoundation.org/UA/2011/03/UANodeSet.xsd"
     xmlns:uax="http://opcfoundation.org/UA/2008/02/Types.xsd"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://
    www.w3.org/2001/XMLSchema-instance">
  <Aliases>
    <Alias Alias="Boolean">i=1</Alias>
    <Alias Alias="Float">i=10</Alias>
    <Alias Alias="Organizes">i=35</Alias>
    <Alias Alias="HasTypeDefinition">i=40</Alias>
    <Alias Alias="HasComponent">i=47</Alias>
  </Aliases>
  <NamespaceUris />
  <UAObject BrowseName="0:Cuve" NodeId="ns=5;s=Cuve" ParentNodeId="i
    =85">
    <DisplayName>BaseObjectType</DisplayName>
    <Description>The base type for all object nodes.</Description>
    <References>
      <Reference IsForward="false" ReferenceType="Organizes">i=85</
    Reference>
      <Reference ReferenceType="HasTypeDefinition">i=58</Reference>
      <Reference ReferenceType="HasComponent">ns=5;s=Cuve.Niveau</
    Reference>
      <Reference ReferenceType="HasComponent">ns=5;s=Cuve.Acidite</
    Reference>
      <Reference ReferenceType="HasComponent">ns=5;s=Cuve.
    Temperature</Reference>
    </References>
  </UAObject>
  <UAVariable BrowseName="0:Niveau" DataType="Boolean" NodeId="ns=5;
    s=Cuve.Niveau" ParentNodeId="ns=5;s=Cuve">
    <DisplayName>Niveau</DisplayName>
    <Description>Niveau</Description>
```

```
    <Value>
      <uax:Boolean>False</uax:Boolean>
    </Value>
    <References>
      <Reference IsForward="false" ReferenceType="HasComponent">ns
    =5;s=Cuve</Reference>
      <Reference ReferenceType="HasTypeDefinition">i=63</Reference>
    </References>
  </UAVariable>
  <UAVariable BrowseName="0:Acidite" DataType="Float" NodeId="ns=5;s
    =Cuve.Acidite" ParentNodeId="ns=5;s=Cuve">
    <DisplayName>Acidite</DisplayName>
    <Description>Acidite</Description>
    <Value>
      <uax:Float>9.99</uax:Float>
    </Value>
    <References>
      <Reference IsForward="false" ReferenceType="HasComponent">ns
    =5;s=Cuve</Reference>
      <Reference ReferenceType="HasTypeDefinition">i=63</Reference>
    </References>
  </UAVariable>
  <UAVariable BrowseName="0:Temperature" DataType="Float" NodeId="ns
    =5;s=Cuve.Temperature" ParentNodeId="ns=5;s=Cuve">
    <DisplayName>Temperature</DisplayName>
    <Description>Temperature</Description>
    <Value>
      <uax:Float>9.99</uax:Float>
    </Value>
    <References>
      <Reference IsForward="false" ReferenceType="HasComponent">ns
    =5;s=Cuve</Reference>
      <Reference ReferenceType="HasTypeDefinition">i=63</Reference>
    </References>
  </UAVariable>
</UANodeSet>
```

Listing B.1 – Espace d'adressage OPC-UA pour l'exemple de l'attaque Maroochy Shire

B.2. Fichier de config. bas niveau pour l'exemple de l'attaque Maroochy Shire

```
## ===================== ##
##       [NETWORKS]      ##
## ===================== ##
Network Server Dev eth0 Addr 10.0.1.254 Mask 24
Network Client Dev eth1 Addr 10.1.1.254 Mask 24


## ===================== ##
##       [PROXIES]       ##
## ===================== ##
Proxy Opcua Port 4840
```

```
## ===================== ##
##       [SERVERS]       ##
## ===================== ##
Server 1 Protocol Opcua Addr 10.0.0.5 Port 4840


## ===================== ##
##       [CLIENTS]       ##
## ===================== ##
Client 1 Addr 10.0.1.5 Session urn:MonApplicationURI


## ===================== ##
##      [CHANNELS]       ##
## ===================== ##
Channel 10 Server 1 Client 1 Access Browse,Read,Write


## ===================== ##
##      [LOCATIONS]      ##
## ===================== ##
Location 1 Server 1 Access Browse,Read Namespace 5 Addr String:Cuve
    Type Void
Location 2 Server 1 Access Browse,Read Namespace 5 Addr String:Cuve.
    Temperature Type Float
Location 3 Server 1 Access Browse,Read Namespace 5 Addr String:Cuve.
    Acidite Type Float
Location 4 Server 1 Access Browse,Read Namespace 5 Addr String:Cuve.
    Niveau Type Bool
Location 5 Server 1 Access Browse,Read,Write Namespace 5 Addr String
    :Pompe.Etat Type Int8


## ===================== ##
##    [ACCESS RULES]     ##
## ===================== ##
Access Browse,Read,Write Channels 10 Locations 5
Access Browse,Read Channels 10 Locations 1-4


## ===================== ##
##      [HISTORIES]      ##
## ===================== ##
History 1 Channel 10 Period 60 MaxAt 1
History 2 Channel 10 Period 60 MaxAt 100


## ===================== ##
##      [MONITORS]       ##
## ===================== ##
Monitor 1 Channel 10 Location 4


## ===================== ##
```

```
##   [LOCATION RULES]    ##
## ==================== ##
Filter Write Location 5 Channel 10 \
        Then Local(4) := Value >= -1; Local(5) := Value <= 1; Local
    (6) := Local(4) && Local(5) \
        If Local(6) Else Reject("Invalid new value for location")
Filter Write Location 5 Channel 10 \
        Then Touch(1) \
        If IsFull(1) Then Reject("Maximum number of access reached
    for location: 5")
Filter Browse,Read,Write Channel 10 Pre \
        Then Touch(2) \
        If IsFull(2) Then Reject("Maximum number of access reached
    for channel: 10")
Filter Write Location 5 Channel 10 \
        Then Local(1) := Monitor(1) == 1; Local(2) := Value == 0;
    Local(3) := Local(1) && Local(2) \
        If Local(3) Then Reject("La pompe ne doit pas être arrêtée !
    ")
```

Listing B.2 – Fichier de config. bas niveau pour l'exemple de l'attaque Maroochy Shire

Table des figures

Liste des tableaux

Liste des listings

Glossaire

A^{2}SPICS	Applicative Attack Scenarios Production for Industrial Control Systems. vii, 37, 38, 115, 116, 123, 125–128, 133, 138, 140, 142, 143, 146, 148–151, 158, 159, 177
AGA	American Gas Association. 14
AKISS	Active Knowledge In Security protocolS. 86, 89
AMDE	Analyse des Modes de Défaillance et de leurs Effets. 119
AMDEC	Analyse des Modes de Défaillance, de leurs Effets et de leur Criticité. 37, 119, 157
ANSSI	Agence Nationale de la Sécurité des Systèmes d'Information. 4, 12, 46, 120, 129, 159
API	Application Programming Interface. vi, 35, 41, 42, 50, 58–60, 63–67, 75, 76, 111, 156
APT	Advanced Persistent Threat. 5
ARAMIS	Architecture Robuste pour les Automates et Matériels des Infrastructures Sensibles. iii, v, vii, 5, 25, 34, 35, 46, 50, 76, 155, 156
ASSI	Analyse de Sécurité de Systèmes Industriels. 111
AVANTSSAR	Automated VAlidatioN of Trust and Security of Service-oriented ARchitectures. 122, 142, 148
AVISPA	Automated Validation of Internet Security Protocols and Applications. 85, 90, 142
BMI	BundesMinisterium des Innern. 13
BSI	Bundesamt für Sicherheit in der Informationstechnik. 13
CL-Atse	Constraint-Logic-based Attack Searcher. 84, 85, 122, 142, 150
CLUSIF	Club de la Sécurité des Systèmes d'Information Français. 120
CPNI	Center for the Protection of National Infrastructure. 13, 14
CPU	Central Processing Unit. 26
CRAMM	CCTA Risk Analysis and Management Method. 120
CSFI	Cyber Security Forum Initiative. 14
CVE	Common Vulnerabilities and Exposures. 111
DCS	Distributed Control System. 6
EBIOS	Expression des Besoins et Identification des Objectifs de Sécurité. 36, 120, 149, 157
ECU	Electronic Control Unit. 18
ENISA	European Network and Information Security Agency. 11, 120
EPROM	Erasable Programmable Read-Only Memory. 26

ERP	Enterprise Resource Planning. 7
FDR	Failures Divergences Refinement. 38, 82, 84, 141, 142
FORTRAN	FORmula TRANslator. 123
HAZOP	HAZard and OPerability study. 36, 119, 157
HLPSL	High-Level Protocol Specification Language. 85
HSM	Hardware Security Module. 14, 35, 62, 84
ICS	Industrial Control System. 4
IDS	Intrusion Detection System. 42
IED	Intelligent Electronic Device. 6
IHM	Interface Homme Machine. 124
IIOT	Industrial Internet of Things. 4
MAC	Message Authentication Code. 86, 97, 99, 100, 105, 107–109
MATLAB	MATrix LABoratory. 122, 123
MEHARI	MÉthode Harmonisée d'Analyse des RIsques. 36, 120, 157
MES	Manufacturing Execution Systems. 6–8, 30, 43, 47, 51, 177
MITM	Man-In-The-Middle. 128, 134, 136
MORDA	Mission Oriented Risk and Design Analysis. 120
NIST	National Institute of Standards and Technology. 12, 14, 120
NRL	United States Naval Research Laboratory. 84
NSA	National Security Agency. 120
OCTAVE	Operationally Critical Threat, Asset, and Vulnerability Evaluation. 120
OFMC	Open-source Fixed-point Model-Checker. 84–87, 112
OSI	Open Systems Interconnection. 32
PAC	Programmable Automation Controler. 6
PLC	Programmable Logical Controler. 6
Prolog	PROgrammation en LOGique. 84
PVS	Prototype Verification System. 86
RAM	Random-Access Memory. 26
ROM	Read-Only Memory. 26
RSSB	Rail Safety and Standards Board. 14
RSSI	Responsables de la Sécurité des Systèmes d'Information. 11
RTU	Remote Terminal Unit. 6, 17, 168
S-CUBE	SCADA Safety and Security. 122
SANS	SysAdmin, Audit, Network and Security. 21
SCADA	Supervisory Control And Data Acquisition. 4, 6–8, 13, 14, 17, 26–30, 34, 35, 42, 43, 45, 47, 51, 52, 121, 124, 156, 169, 177
SCRAM	« Safety Control Rod-Axe Man ». 20

SIEM	Security Information and Event Management. 46
SOC	System On Chip. 35
SPEAR II	Security Protocol Engineering and Analysis Resource. 84, 86
SQUARE	Security QUAlity Requirements Engineering. 120
TA4SP	Tree Automata based on Automatic Approximations for the Analysis of Security Protocols. 84, 85
UML	Unified Modeling Language. 7
VFD	Variable-Frequency Drive. 20, 21
VPN	Virtual Private Network. 46
XML	Extensible Markup Language. 33, 48, 59, 61

Bibliographie

[Abadi and Needham, 1996] Abadi, M. and Needham, R. (1996). Prudent engineering practice for cryptographic protocols. *IEEE transactions on Software Engineering*, 22(1) :6. (cf. p 97).

[Abrams and Weiss, 2008] Abrams, M. and Weiss, J. (2008). Malicious control system cyber security attack case study–maroochy water services, australia. *McLean, VA : The MITRE Corporation*. (cf. p 17, 156 et 169).

[AGA, 2006] AGA (2006). Cryptographic Protection of SCADA Communications. American Gas Association. (cf. p 14).

[Alberts and Dorofee, 2002] Alberts, C. J. and Dorofee, A. (2002). *Managing information security risks : the OCTAVE approach*. Addison-Wesley Longman Publishing Co., Inc. (cf. p 120).

[Allot, 2014] Allot, P. (2014). SCADA et MES : les vérités qui dérangent. *L'usine nouvelle*. (cf. p 7, 8 et 177).

[Alpern and Schneider, 1985] Alpern, B. and Schneider, F. B. (1985). Defining liveness. *Information processing letters*, 21(4) :181–185. (cf. p 43).

[Alpern and Schneider, 1987] Alpern, B. and Schneider, F. B. (1987). Recognizing safety and liveness. *Distributed computing*, 2(3) :117–126. (cf. p 10 et 43).

[Amoah, 2016] Amoah, R. (2016). *Formal security analysis of the DNP3-Secure Authentication Protocol*. PhD thesis, Queensland University of Technology. (cf. p 86, 87 et 112).

[Amoroso, 1994] Amoroso, E. G. (1994). *Fundamentals of computer security technology*. Prentice-Hall, Inc. (cf. p 120).

[ANSSI, 2010] ANSSI (2010). Expression des besoins et identification des objectifs de sécurité. Agence nationale de la sécurité des systèmes d'information. (cf. p 36, 120 et 157).

[ANSSI, 2012a] ANSSI (2012a). Cas pratique. Agence nationale de la sécurité des systèmes d'information. (cf. p 12).

[ANSSI, 2012b] ANSSI (2012b). Maîtriser la ssi pour les systèmes industriels. Agence nationale de la sécurité des systèmes d'information. (cf. p 12).

[ANSSI, 2014a] ANSSI (2014a). Mesures détaillées. Agence nationale de la sécurité des systèmes d'information. (cf. p 12).

[ANSSI, 2014b] ANSSI (2014b). Méthode de classification et mesures principales. Agence nationale de la sécurité des systèmes d'information. (cf. p 12).

[ARAMIS, 2014] ARAMIS (2014). Architecture robuste pour les automates et matériels des infrastructures sensibles (2014-2017). http://aramis.minalogic.net/. (cf. p 34 et 156).

[Arapinis et al., 2014] Arapinis, M., Phillips, J., Ritter, E., and Ryan, M. D. (2014). Statverif : Verification of stateful processes. *Journal of Computer Security*, 22(5) :743–821. (cf. p 147).

[Armando et al., 2012] Armando, A., Arsac, W., Avanesov, T., Barletta, M., Calvi, A., Cappai, A., Carbone, R., Chevalier, Y., Compagna, L., Cuéllar, J., et al. (2012). The avantssar platform for the automated validation of trust and security of service-oriented architectures. *Tools and Algorithms for the Construction and Analysis of Systems*, pages 267–282. (cf. p 122).

[Armando et al., 2005] Armando, A., Basin, D., Boichut, Y., Chevalier, Y., Compagna, L., Cuellar, J., Drielsma, P. H., Heám, P.-C., Kouchnarenko, O., Mantovani, J., Mödersheim, S., von Oheimb, D., R., M., Santiago, J., Turuani, M., Viganò, L., and Vigneron, L. (2005). The AVISPA tool for the automated validation of internet security protocols and applications. In *Proc. of CAV'2005*, LNCS 3576, pages 281–285. (cf. p 85).

[Armando et al., 2014] Armando, A., Carbone, R., and Compagna, L. (2014). Satmc : A sat-based model checker for security-critical systems. In *TACAS*, volume 8413, pages 31–45. (cf. p 84).

[Armando and Compagna, 2005] Armando, A. and Compagna, L. (2005). An optimized intruder model for SAT-based model-checking of security protocols. volume 125, pages 91–108. Elsevier Science Publishers. (cf. p 84).

[Aurora Project, 2014] Aurora Project (2014). Foia response documents. United States Department of Homeland Security. http://s3.documentcloud.org/documents/1212530/14f00304-documents.pdf. (cf. p 18).

[Avizienis et al., 2004] Avizienis, A., Laprie, J.-C., Randell, B., and Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *IEEE transactions on dependable and secure computing*, 1(1) :11–33. (cf. p 10).

[Badrignans et al., 2017] Badrignans, B., Danjean, V., Dumas, J., Elbaz-Vincent, P., Machenaud, S., Orfila, J., Pebay-Peyroula, F., Pebay-Peyroula, F., Potet, M., Puys, M., Richier, J., and Roch, J. (2017). Security Architecture for Embedded Point-to-Points Splitting Protocols. In *WCICSS 2017*. (cf. p 36, 50, 75 et 156).

[Barrett et al., 2009] Barrett, C. W., Sebastiani, R., Seshia, S. A., and Tinelli, C. (2009). Satisfiability modulo theories. *Handbook of satisfiability*, 185 :825–885. (cf. p 37).

[Basin et al., 2003] Basin, D., Mödersheim, S., and Viganò, L. (2003). *Computer Security – ESORICS 2003 : 8th European Symposium on Research in Computer Security, Gjøvik, Norway, October 13-15, 2003. Proceedings*, chapter An On-the-Fly Model-Checker for Security Protocol Analysis, pages 253–270. Springer Berlin Heidelberg, Berlin, Heidelberg. (cf. p 84).

[Bauer et al., 2011] Bauer, A., Leucker, M., and Schallhart, C. (2011). Runtime verification for ltl and tltl. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 20(4) :14. (cf. p 44).

[Bertot and Castéran, 2013] Bertot, Y. and Castéran, P. (2013). *Interactive theorem proving and program development : Coq'Art : the calculus of inductive constructions*. Springer Science & Business Media. (cf. p 37).

[Blanchet, 2001] Blanchet, B. (2001). An efficient cryptographic protocol verifier based on Prolog rules. In *Proceedings of the 14th IEEE Workshop on Computer Security Foundations*, CSFW '01, pages 82–, Washington, DC, USA. IEEE Computer Society. (cf. p 85, 94 et 142).

[Blanchet and Podelski, 2003] Blanchet, B. and Podelski, A. (2003). Verification of cryptographic protocols : Tagging enforces termination. In *Foundations of Software Science and Computation Structures*, pages 136–152. Springer. (cf. p 90).

[Blanchet et al., 2017] Blanchet, B., Smyth, B., and Cheval, V. (2017). Proverif 1.96 : Automatic cryptographic protocol verifier, user manual and tutorial. (cf. p 85, 90, 94 et 142).

[Blankenship, 1986] Blankenship, L. (1986). The conscience of a hacker. *Phrack, Volume One*, (7). (cf. p 15).

[Boichut et al., 2004] Boichut, Y., Héam, P.-C., Kouchnarenko, O., and Oehl, F. (2004). Improvements on the genet and klay technique to automatically verify security protocols. In *Proc. AVIS*, volume 4. (cf. p 84).

[Bortolozzo et al., 2010] Bortolozzo, M., Centenaro, M., Focardi, R., and Steel, G. (2010). Attacking and fixing pkcs# 11 security tokens. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 260–269. ACM. (cf. p 84).

[Broadfoot et al., 2000] Broadfoot, P. J., Lowe, G., and Roscoe, A. (2000). Automating data independence. In *ESORICS*, pages 175–190. Springer. (cf. p 90).

[BSI, 2009] BSI (2009). Cip implementation plan of the national plan for information infrastructure protection. Federal Ministry of the Interior and Bundesamt für Sicherheit in der Informationstechnik. (cf. p 13).

[Buckshaw et al., 2005] Buckshaw, D. L., Parnell, G. S., Unkenholz, W. L., Parks, D. L., Wallner, J. M., and Saydjari, O. S. (2005). Mission oriented risk and design analysis of critical information systems. *Military Operations Research*, 10(2) :19–38. (cf. p 120).

[Burrows et al., 1989] Burrows, M., Abadi, M., and Needham, R. M. (1989). A logic of authentication. In *Proceedings of the Royal Society of London A : Mathematical, Physical and Engineering Sciences*, volume 426, pages 233–271. The Royal Society. (cf. p 81).

[Bushby, 1997] Bushby, S. T. (1997). BACnet : a standard communication infrastructure for intelligent buildings. *Automation in Construction*, 6(5) :529–540. (cf. p 7).

[Byres et al., 2004] Byres, E. J., Franz, M., and Miller, D. (2004). The use of attack trees in assessing vulnerabilities in scada systems. In *Proceedings of the international infrastructure survivability workshop*. (cf. p 121 et 149).

[Byres et al., 2006] Byres, E. J., Hoffman, D., and Kube, N. (2006). On shaky ground–a study of security vulnerabilities in control protocols. *Proc. 5th American Nuclear Society Int. Mtg. on Nuclear Plant Instrumentation, Controls, and HMI Technology*. (cf. p 121).

[Cárdenas et al., 2011] Cárdenas, A. A., Amin, S., Lin, Z.-S., Huang, Y.-L., Huang, C.-Y., and Sastry, S. (2011). Attacks against process control systems : risk assessment,

detection, and response. In *Proceedings of the 6th ACM symposium on information, computer and communications security*, pages 355–366. ACM. (cf. p 44, 75 et 122).

[Chang et al., 1993] Chang, E., Manna, Z., and Pnueli, A. (1993). The safety-progress classification. In *Logic and Algebra of Specification*, pages 143–202. Springer. (cf. p 43 et 76).

[Chen and Abdelwahed, 2014] Chen, Q. and Abdelwahed, S. (2014). A model-based approach to self-protection in scada systems. In *9th International Workshop on Feedback Computing (Feedback Computing 14)*, Philadelphia, PA. USENIX Association. (cf. p 44 et 75).

[Cherdantseva et al., 2015] Cherdantseva, Y., Burnap, P., Blyth, A., Eden, P., Jones, K., Soulsby, H., and Stoddart, K. (2015). A review of cyber security risk assessment methods for SCADA systems. *Computers & Security*, 56 :1 – 27. (cf. p 121 et 149).

[Cheung et al., 2007] Cheung, S., Dutertre, B., Fong, M., Lindqvist, U., Skinner, K., and Valdes, A. (2007). Using model-based intrusion detection for scada networks. In *Proceedings of the SCADA security scientific symposium*, volume 46, pages 1–12. Citeseer. (cf. p 44).

[Cimpean et al., 2012] Cimpean, D., Cano Bernaldo de Quirós, P., and García Gutiérrez, F. (2012). Appropriate security measures for smart grids - Guidelines to assess the sophistication of security measures implementation. Technical report, Technical report, European Union Agency for Network and Information Security (ENISA). (cf. p 12).

[CISHEC, 1977] CISHEC, A. (1977). Guide to hazard and operability studies. *The Chemical Industry Safety and Health Council of the Chemical Industries Association Ltd.* (cf. p 119).

[Clarke et al., 2004] Clarke, G. R., Reynders, D., and Wright, E. (2004). *Practical modern SCADA protocols : DNP3, 60870.5 and related systems*. Newnes. (cf. p 7, 17, 86, 87, 112 et 168).

[Clavel et al., 1996] Clavel, M., Eker, S., Lincoln, P., and Meseguer, J. (1996). Principles of maude. *Electronic Notes in Theoretical Computer Science*, 4 :65–89. (cf. p 85).

[Clinton, 1998] Clinton, W. (1998). Presidential decision directive 63. *The White House, Washington, DC (fas. org/irp/offdocs/pdd/pdd-63. htm)*. (cf. p 11).

[CLUSIF, 2010] CLUSIF (2010). Méthode harmonisée d'analyse des risques. (cf. p 36, 120 et 157).

[Conchon and Caire, 2015] Conchon, S. and Caire, J. (2015). Expression des besoins et identification des objectifs de résilience. *C&esar'15*. (cf. p 130).

[Cortier, 2005] Cortier, V. (2005). Vérifier les protocoles cryptographiques. *Technique et Science Informatiques*, 24(1) :115–140. (cf. p 90).

[Cox, 2011] Cox, D. P. (2011). *The application of autonomic computing for the protection of industrial control systems*. PhD thesis. (cf. p 44 et 75).

[CPNI, 2008] CPNI (2008). Good practice guide – process control and scada secguide. Technical report, Centre for the Protection of National Infrastructure. (cf. p 13).

[CPNI, 2011] CPNI (2011). Good practice guide – cyber security assessments of industrial control systems. Technical report, Centre for the Protection of National Infrastructure. (cf. p 13).

[Cremers, 2008] Cremers, C. (2008). The Scyther Tool : Verification, falsification, and analysis of security protocols. In *Computer Aided Verification, 20th International Conference, CAV*, volume 5123/2008 of *LNCS*, pages 414–418. Springer. (cf. p 85).

[Cremers, 2011] Cremers, C. (2011). Key exchange in ipsec revisited : Formal analysis of ikev1 and ikev2. *Computer Security–ESORICS 2011*, pages 315–334. (cf. p 85).

[CSFI, 2015] CSFI (2015). Csfi atc (air traffic control) cyber security project. Cyber Security Forum Initiative. (cf. p 14).

[Debar et al., 2000] Debar, H., Dacier, M., and Wespi, A. (2000). A revised taxonomy for intrusion-detection systems. *Annales Des Télécommunications*, 55(7) :361–378. (cf. p 44).

[Diallo and Feuillet, 2014] Diallo, D. and Feuillet, M. (2014). Détection d'intrusion dans les systèmes industriels : Suricata et le cas MODBUS. *C&ESAR2014*. (cf. p 44).

[Dierks and Rescorla, 2008] Dierks, T. and Rescorla, E. (2008). The transport layer security (TLS) protocol, version 1.2. IETF RFC 5246. (cf. p 95).

[DIN-19245, 1991] DIN-19245 (1991). Translation of the german national standard din 19245 parts 1 and 2. *Profibus Nutzerorganization eV Std*. (cf. p 7).

[Dittrich et al., 1995] Dittrich, K. R., Gatziu, S., and Geppert, A. (1995). The active database management system manifesto : A rulebase of adbms features. In *International Workshop on Rules in Database Systems*, pages 1–17. Springer. (cf. p 52).

[Dolev and Yao, 1981] Dolev, D. and Yao, A. C. (1981). On the security of public key protocols. *Information Theory, IEEE Transactions on*, 29(2) :198–208. (cf. p 38, 83 et 87).

[Dreier et al., 2017a] Dreier, J., Blot, E., and Lafourcade, P. (2017a). Formal analysis of combinations of secure protocols. In *10th International Symposium on Foundations & Practice of Security*. (cf. p 113).

[Dreier et al., 2017b] Dreier, J., Puys, M., Potet, M.-L., Lafourcade, P., and Roch, J.-L. (2017b). Formally verifying flow integrity properties in industrial systems. In *SECRYPT 2017 - 14th International Conference on Security and Cryptography*, page 12, Madrid, Spain. (cf. p 36, 87, 100, 105, 111, 112 et 157).

[Dumas et al., 2016] Dumas, J., Lafourcade, P., Orfila, J., and Puys, M. (2016). Private multi-party matrix multiplication and trust computations. In *Proceedings of the 13th International Joint Conference on e-Business and Telecommunications (ICETE 2016) - Volume 4 : SECRYPT, Lisbon, Portugal, July 26-28, 2016.*, pages 61–72. Best Paper Award. (cf. p 91).

[Dumas et al., 2017] Dumas, J.-G., Lafourcade, P., Orfila, J.-B., and Puys, M. (2017). Dual protocols for private multi-party matrix multiplication and trust computations. *Computers & Security*, pages –. (cf. p 91).

[Dutertre, 2007] Dutertre, B. (2007). Formal modeling and analysis of the MODBUS protocol. In *Critical Infrastructure Protection*, pages 189–204. Springer. (cf. p 86).

[Dzung et al., 2005] Dzung, D., Naedele, M., von Hoff, T., and Crevatin, M. (2005). Security for industrial communication systems. *Proceedings of the IEEE*, 93(6) :1152–1177. (cf. p 86, 87 et 112).

[Eilenberg and Tilson, 1974] Eilenberg, S. and Tilson, B. (1974). *Automata, languages, and machines*, volume 76. Academic press New York. (cf. p 43).

[Emerson and Clarke, 1980] Emerson, E. and Clarke, E. (1980). Characterizing correctness properties of parallel programs using fixpoints. *Automata, Languages and Programming*, pages 169–181. (cf. p 37).

[Ericson, 1999] Ericson, C. A. (1999). Fault tree analysis. In *System Safety Conference, Orlando, Florida*, pages 1–9. (cf. p 120).

[Falcone et al., 2012] Falcone, Y., Fernandez, J., and Mounier, L. (2012). What can you verify and enforce at runtime ? *STTT*, 14(3) :349–382. (cf. p 77).

[Falcone et al., 2008] Falcone, Y., Fernandez, J.-C., and Mounier, L. (2008). Synthesizing enforcement monitors wrt. the safety-progress classification of properties. In *International Conference on Information Systems Security*, pages 41–55. Springer. (cf. p 44 et 77).

[Falcone et al., 2009] Falcone, Y., Fernandez, J.-C., and Mounier, L. (2009). Enforcement monitoring wrt. the safety-progress classification of properties. In *Proceedings of the 2009 ACM symposium on Applied Computing*, pages 593–600. ACM. (cf. p 77).

[Falliere et al., 2011] Falliere, N., Murchu, L. O., and Chien, E. (2011). W32. stuxnet dossier. *White paper, Symantec Corp., Security Response*, 5(6). (cf. p 19).

[Fong, 2004] Fong, P. W. (2004). Access control by tracking shallow execution history. In *Security and Privacy, 2004. Proceedings. 2004 IEEE Symposium on*, pages 43–55. IEEE. (cf. p 44 et 75).

[Ford-Hutchinson, 2005] Ford-Hutchinson, P. (2005). Securing FTP with TLS. IETF RFC 4217. (cf. p 7).

[Foster Jr et al., 2008] Foster Jr, J. S., Gjelde, E., Graham, W. R., Hermann, R. J., Kluepfel, H. M., Lawson, R. L., Soper, G. K., Wood, L. L., and Woodard, J. B. (2008). Report of the commission to assess the threat to the united states from electromagnetic pulse (emp) attack : Critical national infrastructures. Technical report, DTIC Document. (cf. p 21).

[Fovino et al., 2009] Fovino, I., Carcano, A., Masera, M., and Trombetta, A. (2009). Design and implementation of a secure MODBUS protocol. In Palmer, C. and Shenoi, S., editors, *Critical Infrastructure Protection III*, volume 311 of *IFIP Advances in Information and Communication Technology*, pages 83–96. Springer Berlin Heidelberg. (cf. p 80, 86, 87, 106, 107, 108, 111, 112 et 177).

[Genet, 1998] Genet, T. (1998). Decidable approximations of sets of descendants and sets of normal forms. In *RTA*, volume 1379, pages 151–165. Springer. (cf. p 84).

[Gibson-Robinson et al., 2014] Gibson-Robinson, T., Armstrong, P., Boulgakov, A., and Roscoe, A. W. (2014). Fdr3—a modern refinement checker for csp. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 187–201. Springer. (cf. p 38 et 82).

[Girard, 1987] Girard, J.-Y. (1987). Linear logic. *Theoretical computer science*, 50(1) :1–101. (cf. p 143).

[Graham and Patel, 2004] Graham, J. H. and Patel, S. C. (2004). Security considerations in SCADA communication protocols. Technical report, University of Louisville. (cf. p 87 et 112).

[Greenberg, 2013] Greenberg, A. (2013). Hackers reveal nasty new car attacks—with me behind the wheel. *Forbes*. http://www.forbes.com/sites/andygreenberg/2013/07/24/hackers-reveal-nasty-new-car-attacks-with-me-behind-the-wheel-video. (cf. p 18).

[Greenberg, 2015] Greenberg, A. (2015). Hackers remotely kill a jeep on the highway—with me in it. *Wired*, 7 :21. https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/. (cf. p 18).

[Guiochet and Powell, 2005] Guiochet, J. and Powell, D. (2005). *Etude et analyse de différents dispositifs externes de sécurité-innocuité de type safety bag*. PhD thesis, LAAS-CNRS. (cf. p 44 et 77).

[Hardy, 2012] Hardy, T. L. (2012). *Software and System Safety*. AuthorHouse. (cf. p 20).

[Hayden et al., 2014] Hayden, E., Assante, M., and Conway, T. (2014). An abbreviated history of automation & industrial controls systems and cybersecurity. *SANS Analyst Whitepaper*. (cf. p 16 et 21).

[Hayes and El-Khatib, 2013] Hayes, G. and El-Khatib, K. (2013). Securing MODBUS transactions using hash-based message authentication codes and stream transmission control protocol. In *Communications and Information Technology (ICCIT), 2013 Third International Conference on*, pages 179–184. (cf. p 80, 86, 87, 106, 107, 108, 109, 111, 112 et 177).

[Holt and Kilger, 2012] Holt, T. J. and Kilger, M. (2012). Know your enemy : The social dynamics of hacking. *The Honeynet Project*, pages 1–17. (cf. p 15).

[Horn, 1951] Horn, A. (1951). On sentences which are true of direct unions of algebras. *The Journal of Symbolic Logic*, 16(1) :14–21. (cf. p 85 et 142).

[IEC-31010, 2009] IEC-31010 (2009). Gestion des risques – Techniques d'évaluation des risques. International Electrotechnical Commission. (cf. p 118).

[IEC-60300, 2003] IEC-60300 (2003). Dependability management - Part 3-1 : Application guide - Analysis techniques for dependability - Guide on methodology. International Electrotechnical Commission. (cf. p 118).

[IEC-60812, 1985] IEC-60812 (1985). Analysis techniques for system reliability - Procedure for failure mode and effects analysis (FMEA). International Electrotechnical Commission. (cf. p 37, 119 et 157).

[IEC-61131-3, 2013] IEC-61131-3 (2013). Programmable controllers - Part 3 : Programming languages. International Electrotechnical Commission. (cf. p 26).

[IEC-61158, 2014] IEC-61158 (2014). Industrial communication networks - Fieldbus specifications. International Electrotechnical Commission. (cf. p 7).

[IEC-61508, 2010] IEC-61508 (2010). Functional safety of electrical/electronic/programmable electronic safety-related systems. International Electrotechnical Commission. (cf. p 119).

[IEC-61882, 2001] IEC-61882 (2001). Hazard and operability studies (HAZOP studies). International Electrotechnical Commission. (cf. p 37, 119 et 157).

[IEC-62264, 2013] IEC-62264 (2013). Intégration des systèmes entreprise-contrôle. International Electrotechnical Commission. (cf. p 7).

[IEC-62351, 2016] IEC-62351 (2016). Power systems management and associated information exchange - data and communications security. International Electrotechnical Commission. (cf. p 13).

[IEC-62439, 2016] IEC-62439 (2016). Industrial communication networks - High availability automation networks - Part 3 : Parallel Redundancy Protocol (PRP) and High-availability Seamless Redundancy (HSR). International Electrotechnical Commission. (cf. p 102).

[IEC-62443, 2010] IEC-62443 (2010). Industrial communication networks - Network and system security. International Electrotechnical Commission. (cf. p 11).

[IEC-62541, 2015] IEC-62541 (2015). OPC Unified Architecture. International Electrotechnical Commission. (cf. p 7, 12, 32, 33, 87, 92, 95, 108 et 112).

[IEC-62645, 2014] IEC-62645 (2014). Nuclear power plants - instrumentation and control systems - requirements for security programmes for computer-based systems. International Electrotechnical Commission. (cf. p 13).

[IEC-62859, 2016] IEC-62859 (2016). Nuclear power plants - instrumentation and control systems - requirements for coordinating safety and cybersecurity. International Electrotechnical Commission. (cf. p 13).

[Igure et al., 2006] Igure, V. M., Laughter, S. A., and Williams, R. D. (2006). Security issues in SCADA networks. *Computers & Security*, 25(7) :498 – 506. (cf. p 81).

[ISA99, 2007] ISA99 (2007). Industrial Automation and Control Systems Security. International Society of Automation. (cf. p 11).

[ISO-27000, 2005] ISO-27000 (2005). Information technology — Security techniques — Information security management systems. International Organization for Standardization. (cf. p 8 et 11).

[ISO-27002, 2005] ISO-27002 (2005). Information technology – Security techniques – Code of practice for information security management. International Organization for Standardization. (cf. p 11).

[ISO-27002, 2011] ISO-27002 (2011). Information technology – Security techniques – Information security risk management. International Organization for Standardization. (cf. p 119).

[ISO-27019, 2013] ISO-27019 (2013). Information technology – Security techniques – Information security management guidelines based on ISO/IEC 27002 for process control systems specific to the energy utility industry. International Organization for Standardization. (cf. p 11).

[ISO-8802, 1998] ISO-8802 (1998). Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements. International Organization for Standardization. (cf. p 7).

[Kabir-Querrec, 2017] Kabir-Querrec, M. (2017). *Cyber security of the smart grid control systems : intrusion detection in IEC 61850 communication networks*. PhD thesis, University Grenoble Alpes. (cf. p 118 et 123).

[Khaled, 2017] Khaled, A. (2017). Production des scénarios d'attaques à partir de spécifications. Master's thesis, Université de technologie de Troyes. (cf. p 132).

[Klay and Vigneron, 2009] Klay, F. and Vigneron, L. (2009). Automatic methods for analyzing non-repudiation protocols with an active intruder. In Degano, P., Guttman, J., and Martinelli, F., editors, *Formal Aspects in Security and Trust*, volume 5491 of *Lecture Notes in Computer Science*, pages 192–209. Springer Berlin Heidelberg. (cf. p 113).

[Kleene, 1952] Kleene, S. C. (1952). *Introduction to metamathematics*. (cf. p 57).

[Kletz, 1999] Kletz, T. A. (1999). *HAZOP and HAZAN : identifying and assessing process industry hazards*. IChemE. (cf. p 37 et 157).

[Knowles et al., 2015] Knowles, W., Prince, D., Hutchison, D., Disso, J. F. P., and Jones, K. (2015). A survey of cyber security management in industrial control systems. *International journal of critical infrastructure protection*, 9 :52–80. (cf. p 118).

[Koucham et al., 2016] Koucham, O., Mocanu, S., Hiet, G., Thiriet, J.-M., and Majorczyk, F. (2016). *Detecting Process-Aware Attacks in Sequential Control Systems*, pages 20–36. Springer International Publishing, Cham. (cf. p 44 et 77).

[Kremer and Raskin, 2001] Kremer, S. and Raskin, J.-F. (2001). A game-based verification of non-repudiation and fair exchange protocols. In *International Conference on Concurrency Theory*, pages 551–565. Springer. (cf. p 89).

[Kriaa et al., 2015a] Kriaa, S., Bouissou, M., and Laarouchi, Y. (2015a). A model based approach for SCADA safety and security joint modelling : S-Cube. In *IET System Safety and Cyber Security*. IET Digital Library. (cf. p 122 et 149).

[Kriaa et al., 2012] Kriaa, S., Bouissou, M., and Piètre-Cambacédès, L. (2012). Modeling the stuxnet attack with bdmp : Towards more formal risk assessments. In *Risk and Security of Internet and Systems (CRiSIS), 2012 7th International Conference on*, pages 1–8. IEEE. (cf. p 122).

[Kriaa et al., 2015b] Kriaa, S., Piètre-Cambacédès, L., Bouissou, M., and Halgand, Y. (2015b). A survey of approaches combining safety and security for industrial control systems. *Reliability Engineering & System Safety*, 139 :156–178. (cf. p 121).

[Lafourcade and Puys, 2015] Lafourcade, P. and Puys, M. (2015). Performance evaluations of cryptographic protocols verification tools dealing with algebraic properties. In *Foundations and Practice of Security - 8th International Symposium, FPS 2015, Clermont-Ferrand, France, October 26-28, 2015, Revised Selected Papers*, pages 137–155. (cf. p 85).

[Lamport, 1977] Lamport, L. (1977). Proving the correctness of multiprocess programs. *IEEE transactions on software engineering*, (2) :125–143. (cf. p 43).

[Langner, 2011] Langner, R. (2011). Stuxnet : Dissecting a cyberwarfare weapon. *Security & Privacy, IEEE*, 9(3) :49–51. (cf. p 19).

[Lee and Brewer, 2010] Lee, A. and Brewer, T. (2010). Guidelines for smart grid cyber security : Vol. 1, smart grid cyber security strategy, architecture, and high-level requirements. *NISTIR 7628*. (cf. p 14).

[Lee et al., 2014] Lee, R. M., Assante, M. J., and Conway, T. (2014). German steel mill cyber attack. *Industrial Control Systems*, 30. (cf. p 22).

[Lee et al., 2016] Lee, R. M., Assante, M. J., and Conway, T. (2016). Analysis of the cyber attack on the ukrainian power grid. *SANS Industrial Control Systems*. (cf. p 22).

[Leszczyna et al., 2011] Leszczyna, R., Egozcue, E., Tarrafeta, L., Villar, V. F., Estremera, R., and Alonso, J. (2011). Protecting industrial control systems - Recommendations for europe and member states. Technical report, Technical report, European Union Agency for Network and Information Security (ENISA). (cf. p 12).

[Ligatti et al., 2005] Ligatti, J., Bauer, L., and Walker, D. (2005). Enforcing non-safety security policies with program monitors. In *European Symposium on Research in Computer Security*, pages 355–373. Springer. (cf. p 44).

[Lions, 1996] Lions, J.-L. (1996). Ariane 5 flight 501 failure. (cf. p 81).

[Lowe, 1995] Lowe, G. (1995). An attack on the needham-schroeder public-key authentication protocol. *Information processing letters*, 56(3) :131–133. (cf. p 82, 83 et 177).

[Lowe, 1996] Lowe, G. (1996). Breaking and fixing the needham-schroeder public-key protocol using fdr. In *International Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, pages 147–166. Springer. (cf. p 82).

[Lowe, 1997] Lowe, G. (1997). A hierarchy of authentication specifications. In *Computer security foundations workshop, 1997. Proceedings., 10th*, pages 31–43. IEEE. (cf. p 89).

[Lowe, 1998] Lowe, G. (1998). Casper : A compiler for the analysis of security protocols. *Journal of computer security*, 6(1, 2) :53–84. (cf. p 38, 84 et 141).

[Mack W. Alford, 1985] Mack W. Alford, Leslie Lamport, G. P. M. (1985). *Basic concepts*, pages 7–43. Springer Berlin Heidelberg, Berlin, Heidelberg. (cf. p 43).

[Mahaffey, 2015] Mahaffey, K. (2015). Hacking a Tesla Model S : What we found and what we learned. Lookout Blog. https://blog.lookout.com/blog/2015/08/07/hacking-a-tesla/. (cf. p 18).

[Markov, 1971] Markov, A. (1971). Extension of the limit theorems of probability theory to a sum of variables connected in a chain. *Reprinted in Appendix B of : R. Howard. Dynamic Probabilistic Systems, volume 1 : Markov Chains*. (cf. p 120).

[Markov, 1906] Markov, A. A. (1906). Rasprostranenie zakona bol'shih chisel na velichiny, zavisyaschie drug ot druga. *Izvestiya Fiziko-matematicheskogo obschestva pri Kazanskom universitete*, 15(135-156) :18. (cf. p 120).

[McQueen et al., 2006] McQueen, M. A., Boyer, W. F., Flynn, M. A., and Beitel, G. A. (2006). Quantitative cyber risk reduction estimation methodology for a small scada control system. In *System Sciences, 2006. HICSS'06. Proceedings of the 39th Annual Hawaii International Conference on*, volume 9, pages 226–226. IEEE. (cf. p 121).

[Mead and Stehney, 2005] Mead, N. R. and Stehney, T. (2005). *Security quality requirements engineering (SQUARE) methodology*, volume 30. ACM. (cf. p 120).

[Meadows, 1996] Meadows, C. (1996). The nrl protocol analyzer : An overview. *The Journal of Logic Programming*, 26(2) :113–131. (cf. p 84).

[Meier et al., 2013] Meier, S., Schmidt, B., Cremers, C., and Basin, D. (2013). The tamarin prover for the symbolic analysis of security protocols. In Sharygina, N. and Veith, H., editors, *Computer Aided Verification*, volume 8044 of *Lecture Notes in Computer Science*, pages 696–701. Springer Berlin Heidelberg. (cf. p 85 et 147).

[Milner et al., 1992] Milner, R., Parrow, J., and Walker, D. (1992). A calculus of mobile processes, i. *Information and computation*, 100(1) :1–40. (cf. p 85 et 142).

[MODBUS, 2004] MODBUS (2004). MODBUS IDA, MODBUS messaging on TCP/IP implementation guide v1.0a. (cf. p 7, 31, 107 et 108).

[Morris et al., 2013] Morris, T. H., Jones, B. A., Vaughn, R. B., and Dandass, Y. S. (2013). Deterministic intrusion detection rules for modbus protocols. In *System Sciences (HICSS), 2013 46th Hawaii International Conference on*, pages 1773–1781. IEEE. (cf. p 44).

[Murata, 1989] Murata, T. (1989). Petri nets : Properties, analysis and applications. *Proceedings of the IEEE*, 77(4) :541–580. (cf. p 37).

[Nardella, 2016] Nardella, D. (2016). Snap7 v1.4.1 - reference manual. http://snap7.sourceforge.net/. (cf. p 7).

[Needham and Schroeder, 1978] Needham, R. M. and Schroeder, M. D. (1978). Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12) :993–999. (cf. p 79 et 81).

[Nicely, 1995] Nicely, T. R. (1995). Ten pentium division flaw. In *Virginia Scientists Newsletter*, volume 1, page 3. (cf. p 81).

[NRC, 2007] NRC (2007). Effects of ethernet-based, non-safety related controls on the safe and continued operation of nuclear power stations. *Nuclear Regulatory Commission and others – US NRC Information Notice*, 15. (cf. p 20).

[Patel et al., 2009] Patel, S. C., Bhatt, G. D., and Graham, J. H. (2009). Improving the cyber security of SCADA communication networks. *Commun. ACM*, 52(7) :139–142. (cf. p 81).

[Patel et al., 2008] Patel, S. C., Graham, J. H., and Ralston, P. A. (2008). Quantitatively assessing the vulnerability of critical information systems : A new method for evaluating security enhancements. *International Journal of Information Management*, 28(6) :483–491. (cf. p 121).

[Patel and Yu, 2007] Patel, S. C. and Yu, Y. (2007). Analysis of SCADA security models. *International Management Review*, 3(2) :68. (cf. p 86, 87 et 112).

[Paulson, 1994] Paulson, L. C. (1994). *Isabelle : A generic theorem prover*, volume 828. Springer Science & Business Media. (cf. p 37).

[Pauna and Moulinos, 2013] Pauna, A. and Moulinos, K. (2013). Window of exposure... a real problem for SCADA Systems. *European Union Agency for Network and Information Security*, page 1. (cf. p 12).

[Paxson, 1999] Paxson, V. (1999). Bro : a system for detecting network intruders in real-time. *Computer networks*, 31(23) :2435–2463. (cf. p 44).

[Pearl, 1985] Pearl, J. (1985). Bayesian networks : A model of self-activated memory for evidential reasoning. In *Proceedings of the 7th Conference of the Cognitive Science Society, 1985*, pages 329–334. (cf. p 120).

[Petri, 1962] Petri, C. A. (1962). *Kommunikation mit automaten*. PhD thesis, Technische Hochschule Darmstadt. (cf. p 120).

[Piètre-Cambacédès, 2010] Piètre-Cambacédès, L. (2010). *The relationships between safety and security*. Theses, Télécom ParisTech. (cf. p 10 et 118).

[Piètre-Cambacédès and Bouissou, 2013] Piètre-Cambacédès, L. and Bouissou, M. (2013). Cross-fertilization between safety and security engineering. *Reliability Engineering & System Safety*, 110 :110–126. (cf. p 121).

[Piètre-Cambacédès and Chaudet, 2010] Piètre-Cambacédès, L. and Chaudet, C. (2010). The sema referential framework : Avoiding ambiguities in the terms "security" and "safety". *International Journal of Critical Infrastructure Protection*, 3(2) :55–66. (cf. p 11).

[Piètre-Cambacédès et al., 2011] Piètre-Cambacédès, L., Deflesselle, Y., and Bouissou, M. (2011). Security modeling with bdmp : from theory to implementation. In *Network and Information Systems Security (SAR-SSI), 2011 Conference on*, pages 1–8. IEEE. (cf. p 122).

[Pnueli and Zaks, 2006] Pnueli, A. and Zaks, A. (2006). Psl model checking and run-time verification via testers. In *International Symposium on Formal Methods*, pages 573–586. Springer. (cf. p 44).

[Post et al., 2009] Post, O., Seppälä, J., and Koivisto, H. (2009). The performance of opc-ua security model at field device level. In *ICINCO-RA*, pages 337–341. (cf. p 32).

[Postel and Reynolds, 1985] Postel, J. and Reynolds, J. (1985). File Transfert Protocol. IETF RFC 959. (cf. p 7).

[Puys et al., 2016a] Puys, M., Potet, M., and Lafourcade, P. (2016a). Formal analysis of security properties on the OPC-UA SCADA protocol. In *Computer Safety, Reliability, and Security - 35th International Conference, SAFECOMP 2016, Trondheim, Norway, September 21-23, 2016, Proceedings*, pages 67–75. (cf. p 36, 86, 87, 91, 111, 112 et 157).

[Puys et al., 2016b] Puys, M., Potet, M., and Roch, J. (2016b). Génération systématique de scénarios d'attaques contre des systèmes industriels. In *Approches Formelles dans l'Assistance au Développement de Logiciels, AFADL 2016, Besançon, France*. (cf. p 37, 123, 148 et 158).

[Puys et al., 2017] Puys, M., Potet, M.-L., and Khaled, A. (2017). Generation of applicative attacks scenarios against industrial systems. In *Foundations and Practice of Security - 10th International Symposium, FPS 2017, Nancy, France, October 23-25, 2017, Revised Selected Papers*, pages 137–155. (cf. p 123, 133 et 148).

[Puys et al., 2016c] Puys, M., Roch, J., and Potet, M. (2016c). Domain specific stateful filtering with worst-case bandwidth. In *CRITIS'16, Paris, France*. (cf. p 36, 50, 75, 156 et 167).

[Queille and Sifakis, 1982] Queille, J. and Sifakis, J. (1982). Specification and verification of concurrent systems in cesar. In *International Symposium on programming*, pages 337–351. Springer. (cf. p 37).

[Rescorla, 2000] Rescorla, E. (2000). HTTP Over TLS. IETF RFC 2818. (cf. p 7).

[Rocchetto and Tippenhauer, 2016] Rocchetto, M. and Tippenhauer, N. O. (2016). Cpdy : Extending the dolev-yao attacker with physical-layer interactions. In *International Conference on Formal Engineering Methods*, pages 175–192. Springer. (cf. p 122 et 150).

[Rocchetto and Tippenhauer, 2017] Rocchetto, M. and Tippenhauer, N. O. (2017). Towards formal security analysis of industrial control systems. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 114–126. ACM. (cf. p 122, 142 et 150).

[Roscoe, 1994] Roscoe, A. W. (1994). *A classical mind : essays in honour of CAR Hoare*. Prentice Hall International (UK) Ltd. (cf. p 38 et 82).

[Ross et al., 2013] Ross, R., Stoneburner, G., Graubart, R., Dempsey, K., Porter, E., Hodge, B., Quigg, K., Enloe, C., Stine, K., Fabius, J., Faigin, D., Johnson, A., Kaiser, L., Miller, P., Miravalle, S., and Pillitteri, V. (2013). Security and privacy controls for federal information systems and organizations. *NIST Special Publication*, 800 :53. (cf. p 13 et 120).

[Roussel, 1975] Roussel, P. (1975). *PROLOG : Manuel de Reference et d'Utilisation*. Université d'Aix-Marseille II. (cf. p 84).

[RSSB, 2016] RSSB (2016). Rail cyber security guidance to industry. Rail Safety and Standards Board. (cf. p 14).

[Saul and Hutchison, 1999] Saul, E. and Hutchison, A. (1999). SPEAR II – the security protocol engineering and analysis resource. (cf. p 84).

[Schmidt et al., 2012] Schmidt, B., Meier, S., Cremers, C., and Basin, D. (2012). Automated analysis of diffie-hellman protocols and advanced security properties. In *Computer Security Foundations Symposium (CSF), 2012 IEEE 25th*, pages 78–94. (cf. p 85 et 147).

[Schneider, 2000] Schneider, F. B. (2000). Enforceable security policies. *ACM Transactions on Information and System Security (TISSEC)*, 3(1) :30–50. (cf. p 43).

[Schneider, 1998] Schneider, S. (1998). Verifying authentication protocols in csp. *Software Engineering, IEEE Transactions on*, 24(9) :741–758. (cf. p 89).

[Schneier, 1999] Schneier, B. (1999). Attack trees. *Dr. Dobb's journal*, 24(12) :21–29. (cf. p 120).

[Sharangpani and Barton, 1994] Sharangpani, H. and Barton, M. (1994). Statistical analysis of floating point flaw in the pentium processor. *Intel Corporation*. (cf. p 81).

[Slay and Miller, 2007] Slay, J. and Miller, M. (2007). Lessons learned from the maroochy water breach. In *International Conference on Critical Infrastructure Protection*, pages 73–82. Springer. (cf. p 17, 156 et 169).

[Snort, 2016] Snort (2016). Snort : Open source network intrusion prevention system. https://www.snort.org. (cf. p 44).

[Stouffer et al., 2011] Stouffer, K., Falco, J., and Karen, S. (2011). Guide to industrial control systems (ICS) security. *NIST special publication*, 800(82). (cf. p 12 et 120).

[Suricata, 2016] Suricata (2016). Suricata : Open Source IDS / IPS / NSM engine. https://suricata-ids.org/. (cf. p 44).

[Ten et al., 2010] Ten, C.-W., Manimaran, G., and Liu, C.-C. (2010). Cybersecurity for critical infrastructures : Attack and defense modeling. *IEEE Transactions on Systems, Man, and Cybernetics-Part A : Systems and Humans*, 40(4) :853–865. (cf. p 122).

[Turuani, 2006] Turuani, M. (2006). The CL-Atse Protocol Analyser. In Frank Pfenning, editor, *17th International Conference on Term Rewriting and Applications - RTA 2006 Lecture Notes in Computer Science*, volume 4098 of *LNCS*, pages 277–286. Springer. (cf. p 84 et 122).

[Villemeur, 1988] Villemeur, A. (1988). Sureté de fonctionnement des systèmes industriels : fiabilité-facteurs humains, informatisation. (cf. p 119).

[Viswanathan and Kim, 2004] Viswanathan, M. and Kim, M. (2004). Foundations for the run-time monitoring of reactive systems–fundamentals of the mac language. In *International Colloquium on Theoretical Aspects of Computing*, pages 543–556. Springer. (cf. p 44).

[Waldner, 1990] Waldner, J.-B. (1990). *CIM : les nouvelles perspectives de la production*. Dunod. (cf. p 5 et 177).

[Wanying et al., 2015] Wanying, Q., Weimin, W., Surong, Z., and Yan, Z. (2015). The study of security issues for the industrial control systems communication protocols. *Joint International Mechanical, Electronic and Information Technology Conference (JIMET 2015)*. (cf. p 86, 87 et 112).

[Weiss, 2010] Weiss, J. (2010). *Protecting industrial control systems from electronic threats*. Momentum Press. (cf. p 16, 17, 18, 20, 21, 156, 168 et 169).

[Williams, 1991] Williams, T. J. (1991). *A Reference Model for Computer Integrated Manufacturing (CIM) : A Description from the Viewpoint of Industrial Automation : Prepared by CIM Reference Model Committee International Purdue Workshop on Industrial Computer Systems*. Instrument Society of America. (cf. p 5 et 177).

[Wool, 2004] Wool, A. (2004). A quantitative study of firewall configuration errors. *Computer*, 37(6) :62–67. (cf. p 59).

Abstract

Industrial systems, also called SCADA (for Supervisory Control And Data Acquisition), are targeted by cyberattacks since Stuxnet in 2010. Due to the criticality of their interaction with the real world, these systems can be really harmful for humans and environment. As industrial systems have historically been physically isolated from the rest of the world, they focused on the protection against outages and human mistakes (also called safety). Cybersecurity differs from safety in the way that an adversary is willing to harm the system and will learn from his mistakes. One of the difficulty in terms of cybersecurity of industrial systems is to make coexist security properties with domain specific constraints. We tackle this question with three main axes.

First, we propose a filter dedicated to industrial communications, allowing to enforce applicative properties. Then, we focus on formal verification of cryptographic protocols applied to industrial protocols such as MODBUS or OPC-UA. Using well-known tools from the domain, we model the protocols in order to check if they provide security properties including confidentiality, authentication and integrity. Finally, we propose an approach named ASPICS (for Applicative Attack Scenarios Production for Industrial Control Systems) to study if safety properties (similar to those verified by our filter) can actually be jeopardized by attackers depending on their position and capacity. We implement this approach in the UPPAAL model-checker and study its results on a proof-of-concept example.

Résumé

Les systèmes industriels, souvent appelés SCADA (pour Système d'acquisition et de contrôle de données) sont la cible d'attaques informatiques depuis Stuxnet en 2010. Dû à la criticité de leurs interactions avec le monde réel, ils peuvent représenter une menace pour l'environnement et les humains. Comme ces systèmes ont par le passé été physiquement isolés du reste du monde, ils ont été majoritairement protégés contre des pannes et des erreurs (ce qu'on appelle la sûreté). La sécurité informatique diffère de la sûreté dans le sens où un attaquant cherchera activement à mettre en défaut le système et gagnera en puissance au cours du temps. L'un des challenges dans le cadre de la sécurité des systèmes industriels est de faire cohabiter des propriétés de sécurité avec les contraintes métier du système. Nous répondons à cette question par trois axes de recherche.

Tout d'abord, nous proposons un filtre dédié aux communications des systèmes industriels, permettant d'exprimer des propriétés au niveau applicatif. Ensuite, nous nous intéressons à la vérification de protocoles cryptographiques appliquée à des protocoles industriels comme MODBUS ou OPC-UA. À l'aide d'outils classiques du domaine, nous modélisons les protocoles afin de vérifier s'ils garantissent des propriété de confidentialité, d'authentification et d'intégrité. Enfin, nous proposons une approche, nommée ASPICS (pour *Applicative Attack Scenarios Production for Industrial Control Systems*), permettant de vérifier si des propriétés de sûreté (similaires à celles vérifiées par le filtre) peuvent être mises en défaut par des attaquants en fonction de leur position et de leur capacité. Nous implémentons cette analyse dans le model-checker UPPAAL et l'appliquons sur un exemple.