Abdelaziz Khaled	Branche: SSI
Responsable pédagogique à UTT:	Année: 2017
Pr. Patrick Lallement	Semèstre: printemps

Université de technologie de Troyes

Production des scénarios d'attaques à partir de spécifications

Récemment, le nombre de cyber attaques de systèmes de contrôle de production industrielle n'a cessé qu'augmenter depuis Stuxnet [15] en 2010. De part leur interaction avec le monde physique, leur protection est devenue une priorité pour les agences gouvernementales. Dans ce rapport, nous proposons une approche modulaire pour évaluer la sécurité des systèmes industriels. Cette approche vise à trouver des attaques applicatives prenant en compte différents paramètres comme le comportement des procédés industriels, les propriétés de sûreté qui doivent être assurées, ainsi que les positions et les capacités des attaquants. Nous instrumentons notre approche à l'aide du model checker UPPAAL, l'appliquons sur un exemple industriel concret et nous montrons comment les propriétés attendues, les topologies de réseau et les attaquants peuvent changer radicalement les résultats obtenus.

Entreprise: Laboratoire Verimag	Mots-clés	
Lieu: Grenoble	Mot clé 1: SCADA	Mot clé 3: Modèle d'attaquant
Responsable: Pr.Marie-Laure Potet	Mot clé 2: Sûreté	Mot clé 4: Model checking

utt université de technologie Troyes

Remerciement

Je tiens à remercier toutes les personnes qui ont contribué au succès de mon stage et qui m'ont aidé lors de la rédaction de ce rapport.

Tout d'abord, je tiens à remercier vivement mon maitre de stage, Pr. Marie-Laure Potet, responsable du l'équipe PACSS au sein de laboratoire VERIMAG, pour son accueil, le temps passé ensemble et le partage de son expertise au quotidien. Grâce aussi à sa confiance, j'ai pu m'accomplir totalement dans mes missions.

Je remercie également Mr. Maxime Pyus qui m'a beaucoup aidé à comprendre le problématique et aussi pour les conseils.

Enfin, je tiens à remercier toutes les personnes qui m'ont conseillé et relu lors de la rédaction de ce rapport de stage : ma famille, mes amis, camarade de promotion.

Table des matières

Table des figures

Liste des tableaux

1 Introduction

Les systèmes industriels sont aujourd'hui fortement informatisés et interconnectés avec les systèmes d'information classiques. À ce titre, ils sont exposés aux mêmes menaces, avec des conséquences potentiellement plus graves. Dans ce rapport nous travaillons spécifiquement sur les systèmes de contrôle de production industrielle. Généralement appelés SCADA, ils contrôlent les processus industriels tels que la production d'électricité, le traitement de l'eau ou le transport. Ètant donné que ces processus sont généralement critiques, tout incident peut potentiellement nuire aux humains et à l'environnement. L'une des attaques les plus annoncées a été Stuxnet en 2010 [15] où un ver a réussi à saboter une installation nucléaire en Iran. Cette attaque a amené les gens à se rendre compte qu'une attaque par ordinateur peut avoir des effets désastreux dans le monde physique.

Des attaques plus récentes contre ces systèmes ont été révélées au cours des dernières années. Par exemple en 2014 contre une aciérie allemande [16] où les attaquants parviennent à prendre le contrôle d'un haut fourneau ou en 2016 en Ukraine [17] provoquant une panne de courant massive en hiver. Les systèmes industriels veulent assurer principalement les propriétés de la disponibilité et l'intégrité, tandis que les systèmes informatiques traditionnels se concentrent souvent sur la confidentialité et l'authentification. De plus, la durée de vie de leurs appareils peut varier entre 20 et 40 ans et ils sont vraiment difficiles à mettre à jour en cas de vulnérabilités.

Les systèmes indusriels communiquent avec des protocoles qui n'utilisent aucune mécanisme de sécurité. Par exemple, MODBUS et DNP3 ne fournissent aucune sécurité, alors qu'un protocole de communication plus récent nommé OPC-UA comprend l'utilisation de la cryptographie [7][22] (mais actuellement rarement utilisé en pratique).

Travaux relatifs. Dans la vérification de la sécurité des systèmes industriels plusieur approches ont été proposé. En 2013 la norme CEI 62443 [12] qui permet de sécuriser les systèmes industriels en divisant en plusieurs zones et en chiffre les communications entre les zones. En 2015 Kriaa et al. [24] décrivent S-CUBE, une approche probabiliste pour inclure les propriétés de serûté aux côtés de la sécurité. En outre, beaucoup d'approches ont été proposées pour modéliser et évaluer la sécurité des systèmes informatiques classiques, y compris : Ekstedt et al. [8], Ou et al. [19] ou Holm et al. [10].

Contributions. Dans ce contexte, nous proposons une approche modulaire pour évaluer la robustesse des systèmes industriels. Notre approche permet de trouver des attaques contre les systèmes en tenant compte de différents paramètres tels que le comportement du processus et les propriétés de sûreté qui doivent être assurer. Nous avons implémenté notre approche à l'aide du model checker UPPAAL pour automatiser la génération des scénarios d'attaques. Nous sommes intéressés à ce que nous appelons les attaques applicatives. Nous supposons qu'un attaquant a déja exploité certains infractions de sécurité pour accéder au système. Nous ciblons particulièrement sur la recherche de quelles actions peut-il effectivement accomplir et quelles en sont les conséquences sur le processus industriel.

Plan. Nous présentons les systèmes SCADA et leur protocoles de communications dans la section 2. Ensuite, dans la section 3 nous présentons un aparçu du model checker UPPAAL. Dans la section 4 nous présentons l'architecture de notre approche et les différents composants, puis dans la section 5 nous appliquons notre approche sur un exemple industriel concret. Enfin, nous concluons ce rapport dans la section 6.

Mission. Mon sujet de stage est une partie des travaux menés dans l'équipe PACSS (projets Aramis, Sacade et thèse de Maxime Puys). Le but du stage était de mettre en place un prototype d'expérimentation. La partie novatrice du sujet était la modélisation d'intrus présentant des capacités d'action et de déduction différentes et la mise en oeuvre de cette approche sur des études de cas, ma mession était :

— Comprendre les problématiques de la sécurisation des systèmes industriels et d'acquérir des connaissances sur les preuves à base d'intrus, telles que pratiquées par les outils de vérification de protocoles de sécurité.
— Proposer des modèles d'intrus.
— Proposer un framework qui permet d'utiliser les modèles d'intrus pour générer les scénarios d'attaques contre les systèmes SCADA.
— Participer à la rédaction d'un papier de recherche qui présente l'architecture mise en place et des résultats d'expérimentation.

Article. Maxime Puys, Marie-Laure Potet and Abdelaziz Khaled. 2017. Generation of Applicative Attacks Scenarios Against Industrial Systems. In Proceedings of ISSA'17, ECSA'17, September 11–15, 2017, Canterbury, UKGB, 7 pages.

2 Les systèmes de supervision et de contrôle SCADA

Le pilotage d'un procédé dans le domaine industriel implique la connaissance, la surveillance et la maîtrise de certains paramètres. Chaque procédé possède ses exigences, et chaque équipement a ses conditions de fonctionnement. De part leur interaction avec le monde physique, ces systèmes peuvent représenter une réelle menace pour leur environnement d'où nécessitent de mise en oeuvre des systèmes pour le contrôle et la sécurité de ces installations afin de respecter les exigences réglementaires.

Dans cette section, nous allons présenter les systèmes SCADA et ses différents protocoles de communication.

2.1 Le système SCADA

SCADA est un acronyme qui signifie le contrôle et la supervision par acquisition de données (en anglais : Supervisory Control and Data Acquisition), est un système qui permet de piloter et de superviser en temps réel et à distance des procédés de production souvent géographiquement très éloignés d'un site central [1][23]. Figure 1 représente l'architecture général d'un système SCADA.

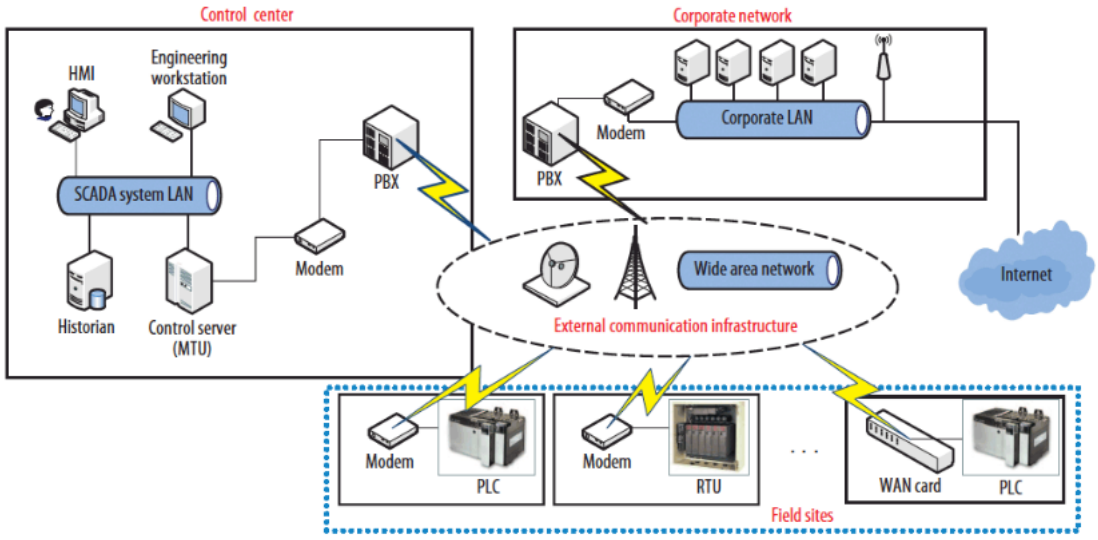


FIGURE 1 – Architecture général d'un système SCADA.

Principalement un système SCADA se compose de [1, 14, 23] :

— Une IHM qui présente les données à un opérateur humain et en charge de superviser le procédé.
— Un RTU (Remote Terminal Unit) : il collecte les informations à partir des équipements du terrain et les transmet au MTU via le système de communication.

— Un MTU (Master Terminal Units) : il collecte les données provenant des RTU, les rend accessibles aux opérateurs via l'HMI et transmet les commandes nécessaires des opérateurs vers les équipements de terrain.
— Un système de communication : moyen de communication entre MTU et les différents RTU, la communication peut être par le biais de l'Internet, réseaux sans fil ou câblé, ou le réseau téléphonique,...

2.2 Protocoles de communication

Les systèmes SCADA utilisent un ensemble de protocoles de communication dédiés, tels que MODBUS, DNP3 et OPC-UA, pour établir la communication entre les différents composants du système. Dans cette section nous présentons deux protocoles, MODBUS et OPC-UA.

2.2.1 Protocole MODBUS

Le protocole MODBUS [11][20] a été développé par Gould Modicon. Ce protocole a rencontré un grand succès depuis sa création en raison de sa bonne fiabilité. On retrouve deux modes de communication dans le protocole MODBUS :

1. *Mode Série :* les messages sont transmis entre un *maître* et un *esclave* à l'aide des modes de transmission ASCII ou RTU, c'est le maître qui doit lire et écrire dans chaque esclave. Le message la est constitué de l'adresse de l'esclave concerné, le PDU et le code de vérification d'erreur. Figure 2 représente l'architecture de MODBUS en mode série.

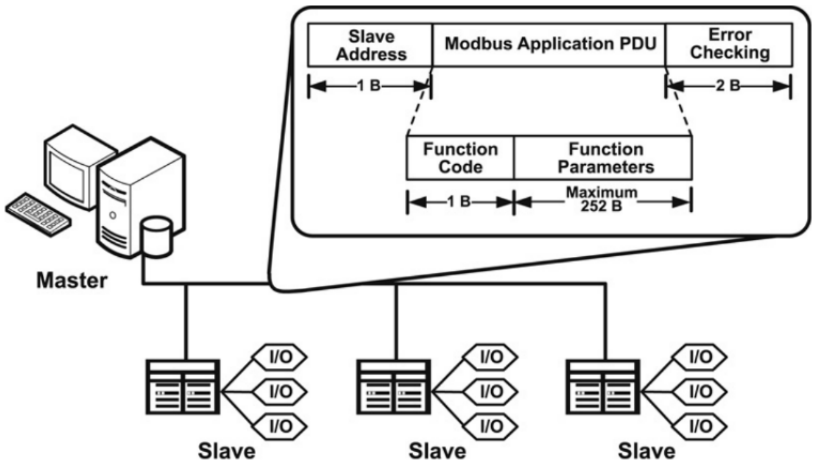


FIGURE 2 – Architecture MODBUS en mode série.

L'adresse de l'esclave dans un message représente la destination ou la source du message. Dans le cas d'une communication par broadcast, message utilise l'adresse de l'esclave *Zero*. Le PDU MODBUS comporte deux champs, un octet pour représenter le type de fonction (lire, écrire) et des paramètres de fonction (maximum 252 octets).

Les messages de réponse ont la même structure que les messages de demande. Il existe deux types de réponse, une réponse *positive* qui informe le maître que l'esclave a effectué avec succès l'action demandée, et une réponse *négative* qui notifie au maître que la demande n'a pas pu être effectuée par l'esclave addressé.

2. *Mode TCP :* fonnctione sur le mode *Client/Serveur*, où chaque client lit et écrit dans le serveur. Les échanges en MODBUS TCP sont similaires à MODBUS série, sauf que les échanges sont encapsulés dans les messages TCP. La Figure 3 montre un maître connecté à plusieurs esclaves via un réseau IP.

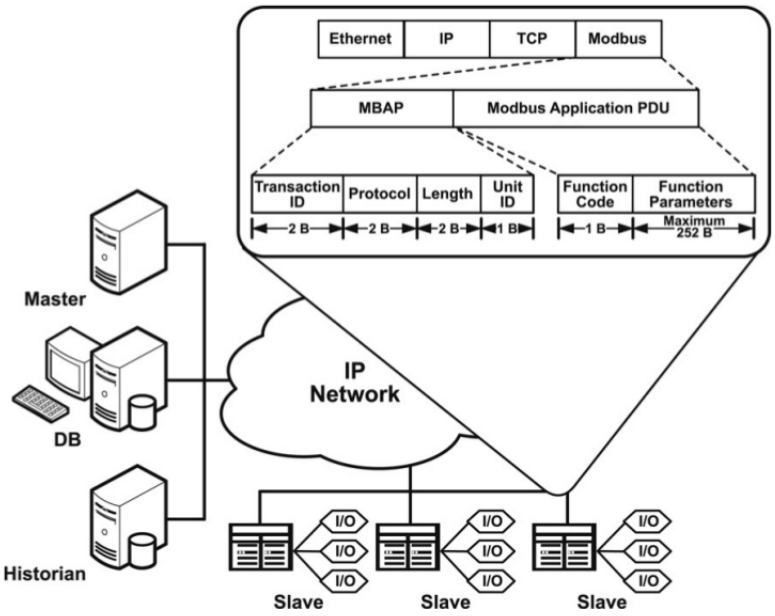


FIGURE 3 – Architecture MODBUS

Dans le mode TCP, l'esclave est désigné comme *serveur* car il exécute l'opération et le maître est désigné comme *client*. Pour la structure des messages, le code de verification d'erreur est inutile en mode TCP. Aussi il y a une autre partie qui s'appelle *MBAP* qui comporte quatre champs : (i) un identificateur de transaction, qui permet d'associer les demandes de correspondance et les réponses sur un canal de communication, (ii) un identificateur de protocole qui indique le protocole d'application encapsulé par l'en-tête MBAP (zéro pour MODBUS), (iii) une longueur qui indique la longueur en octets des champs identificateur d'unité et PDU, (iv) un identifiant de l'unité qui indique l'esclave associé à la transaction.

2.2.2 Protocole OPC-UA

OPC Unified Architecture (OPC-UA) [18] est un protocole de communication machine à machine, développé par la Fondation OPC. Il peut être intégré à n'importe quel système d'expoilation autant en mode filaire que sans fil grâce à sa flexibilité, aussi il peut être utiliser par une

machine à forte capacité ou par de petits objets connectés (IOT). En outre, OPC-UA peut être utilisé avec trois modes de sécurité, nommément *None*, *Sign* et *SignAndEncrypt*.

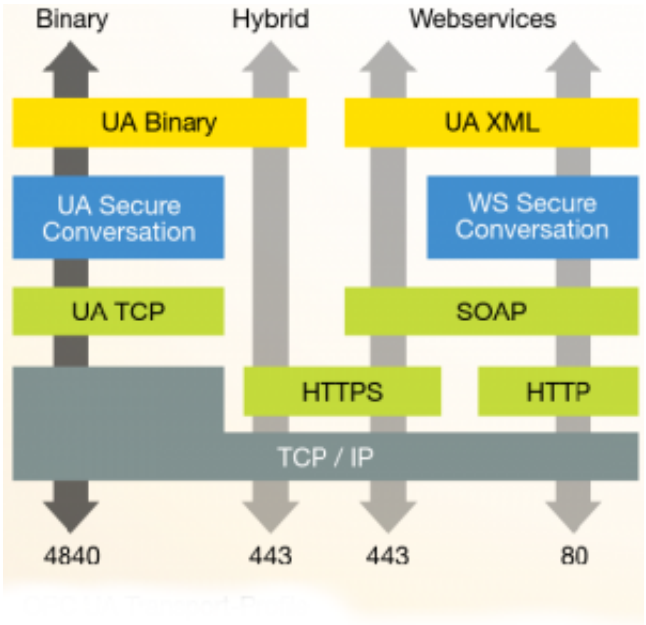


FIGURE 4 – Les modes de transport dans OPC-UA.

Dans OPC-UA on trouve deux modes de transports : (i) mode TCP où les messages sont emballés dans une structure spécifiée par le protocole binaire OPC TCP et la structure est transmise à l'aide d'un socket ou d'un socket sécurisé, (ii) en mode SOAP/HTTP(S) les messages sont emballés dans le corps d'un message SOAP et transmis sur HTTP (S). Un message SOAP est un document XML.

2.3 Classification basée sur la génération

Les systèmes SCADA ont évolué dans 4 générations comme suit [13] :

2.3.1 Première génération : "monolithique"

Dans la première génération, Les réseaux n'existant pas au moment où SCADA a été développé. Ainsi, les systèmes SCADA étaient des systèmes indépendants sans connexion à d'autres systèmes. Les protocoles de communication utilisés sont le plus souvent propriétaires. La première génération de systèmes SCADA suit une architecture redondante car un ordinateur central de secours est connecté au niveau du bus informatique et activé en cas de panne de l'ordinateur central principal.

2.3.2 Deuxième génération : "distribuée"

Dans la deuxième génération, le traitement est distribué sur plusieurs stations qui sont connectées via un réseau local et partagent des informations en temps réel. Chaque station est responsable d'une tâche particulière, ce qui rend la taille et le coût de chaque station inférieurs à celui utilisé dans la première génération. Les protocoles de communication sont encore propriétaires, ce qui peut mener à des problèmes de sécurité importants des systèmes SCADA.

2.3.3 Troisième génération : "en réseau"

Dans la troisième génération, les systèmes SCADA utilisent des standards et des protocoles ouverts, ce qui permet de distribuer des fonctionnalités à travers un WAN plutôt qu'un LAN. En raison de l'utilisation de protocoles standard et du fait que de nombreux systèmes SCADA en réseau sont accessibles depuis Internet, les systèmes sont potentiellement vulnérables aux cyber-attaques. D'autre part, l'utilisation de protocoles standard et de techniques de sécurité signifie que des améliorations de sécurité standard sont applicables aux systèmes SCADA.

2.3.4 Quatrième génération : "Internet des objets"

Dans la quatrième génération, les systèmes SCADA ont adopté les technologies de l'internet des objets pour réduire les coûts d'infrastructure. La maintenance et l'intégration est également très facile pour la quatrième génération par rapport aux systèmes SCADA antérieurs.

3 Le Model Checker UPPAAL

Les méthodes formelles sont des techniques mathématiques permettent d'obtenir une très forte assurance de l'absence de bug dans les logiciels ou matériels électroniques. Il existe plusieurs familles de méthodes formelles tel que : le typage, **le model checker**, la vérification déductive, l'analyse statique. Dans notre travail on utilise le model checker comme méthode pour générer les scénarios d'attaques.

3.1 C'est quoi un model checker ?

Un model checker [2, 5] est une méthode algorithmique qui permet de vérifier automatiquement les systèmes concurrents modélisés par les machines à états finis. Il a été largement utilisé dans la pratique pour vérifier les systèmes embarqués et les protocoles de communication [25]. Initialement, le Model Checker nécessité le modèle du système (désigné par M), la spécification de la propriété Φ à garantir et l'algorithme de vérification qui vérifie si le modèle satisfait ou non la propriété. La Figure 5 représente une aperçu général de l'approche de model checker.

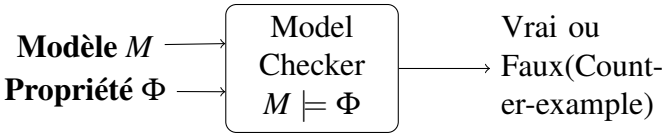


FIGURE 5 – Schéma général de l'approche de model checker.

3.2 L'outil Uppaal

UPPAAL est une boîte à outils pour la validation (via la simulation graphique) et la vérification formelle (via le model checker) des systèmes temps réel [3]. Cet outil a été développé en partenariat avec l'université d'Uppsala et l'université d'Aalborg. Il se compose de deux parties, une interface graphique (GUI) implémentée en JAVA et un moteur de model checker mis en oeuvre en C ++. La Figure 6 représente l'architecture de UPPAAL.

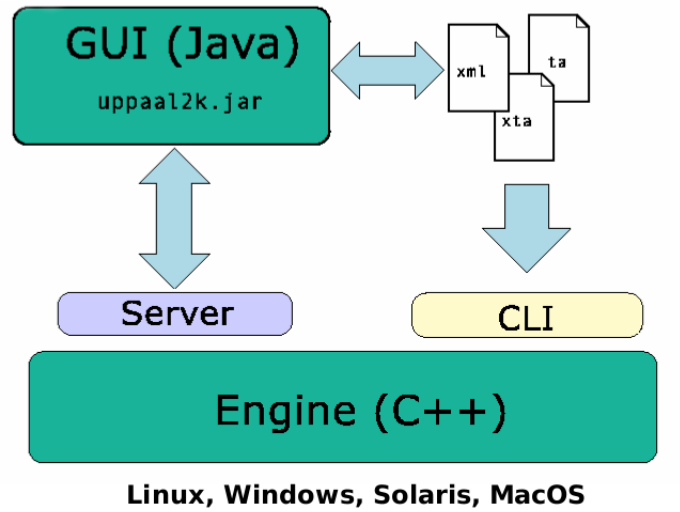


FIGURE 6 – Architecture de UPPAAL.

3.2.1 Modélisation

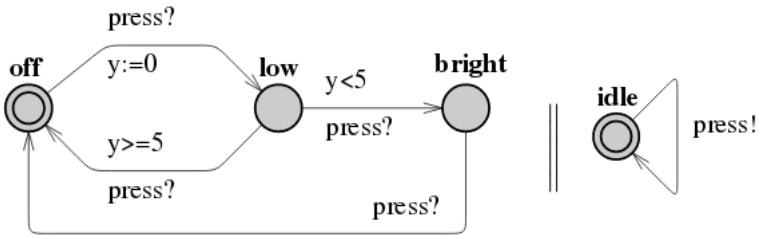
UPPAAL est basé sur les automates temporisés [3]. Un automate temporisé [4] est une machine d'état finie avec des variables d'horloge. Il utilise un modèle de temps dense où une variable d'horloge évalue à un nombre réel. Un système est modélisé comme une composition d'automates.

Dans ce modèle d'automates on différencie trois types d'états : *normal*, *urgent* et *commited*, où chaque état peut comporter une condition sur les horloges, appelée invariant. Une transition de l'automate peut comporter :

— une garde, qui exprime une condition sur les valeurs des variables.

— une synchronisation de la forme *envoyer*! ou *recevoir*?.
— une mise à jour de certaines variables.
— sélectionn d'une valeur à partir d'un intervalle des entiers d'une manière indéterministe.

La Figure 7 montre la modélisation d'une simple lampe par un automate temporisé. La lampe comporte trois états : *off*, *low* et *bright*. Si l'utilisateur appuie sur un bouton (c'est-à-dire, se synchronise avec *press*?), alors la lampe est allumée (low) si l'utilisateur appuie autre fois sur le bouton, la lampe est éteinte. Toutefois, si l'utilisateur appuie rapidement sur le bouton deux fois, la lampe est allumée et devient lumineuse (bright). L'horloge y de la lampe est utilisée pour détecter si l'utilisateur est rapide (y <5) ou lent (y> = 5).



(a) Lamp. (b) User.

FIGURE 7 – Exemple d'une lampe.

Définition 1 (Automate temporisé) un automate temporisé est un $6-uplet(L,l_0,C,A,E,I)$ où : L est un ensemble fini d'etats, l_0 est l'état initial, C est un ensemble fini d'horloges, A est un ensemble fini d'actions, $E \subset L \times A \times B(C) \times 2^C \times L$ un ensemble d'arêtes entre l'états avec une action, une garde et un ensemble d'horloges à réinitialiser et $I : L \longrightarrow B(C)$ attribue des invariants aux états.

3.2.2 Spécification

Pour la spécification des propriétés, UPPAAL utilise une version simplifiée de la logique CTL [2, 5] qui s'exprime par 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 dans tous les états atteignables. $E\diamond\phi$ signifie que parfois ϕ peu être vrai dans certains états. $E\Box\phi$ signifie qu'il y a un chemin où ϕ est toujours vrai. $A\diamond\phi$ signifie que dans chaque chemin se trouve un état où ϕ est vrai, et $\rightarrow$ et $\neg$ sont respectivement l'implication et la négation. La Figure 8 présente les requêtes CTL sous form de graphe.

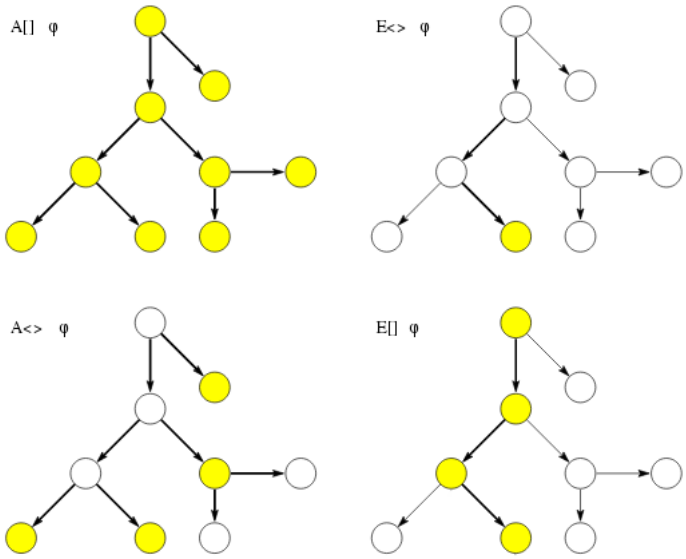


FIGURE 8 – Présentation des requêtes sous forme de graphe.

3.2.3 Le syntaxe du Uppaal

Les expressions dans Uppaal sont similaires à celle du langage C++ [3]. On trouve des variable, des constants, fonctions,...ect. La Figure 9 représente le grammaire UPPAAL en BNF.

Expression → ID | NAT
| *Expression* '[' *Expression* ']'
| '(' *Expression* ')'
| *Expression* '++' | '++' *Expression*
| *Expression* '--' | '--' *Expression*
| *Expression AssignOp Expression*
| *UnaryOp Expression*
| *Expression BinaryOp Expression*
| *Expression* '?' *Expression* ':' *Expression*
| *Expression* '.' ID

UnaryOp → '-' | '!' | 'not'

BinaryOp → '<' | '<=' | '==' | '!=' | '>=' | '>'
| '+' | '-' | '*' | '/' | '%' | '&'
| '|' | '^' | '<<' | '>>' | '&&' | '||'
| '<?' | '>?' | 'and' | 'or' | 'imply'

AssignOp → ':=' | '+=' | '-=' | '*=' | '/=' | '%='
| '|=' | '&=' | '^=' | '<<=' | '>>='

FIGURE 9 – Grammaire de UPPAAL.

4 La génération automatiques des scénarios d'attaques

Dans cette section, nous décrivons comment nous déployons notre approche dans le model checker UPPAAL [3]. Nous montrons d'abord comment modéliser le système. Ensuite, nous détaillons les différents modèles d'attaquant et leur capacité et enfin nous décrivons la spécification des propriétés de sûreté.

4.1 Architecture de framework

La Figure 10 illustre l'architecture globale de notre framework [21]. Il contient trois composants : (i) le modèle du système, (ii) les modèles d'attaquants, et (iii) la spécification des propriétés de sûreté. Plusieurs modèles sont déjà prédéfinis dans la bibliothèque que nous fournissons à l'utilisateur (y compris les clients, les serveurs, les attaquants et les topologies de réseau). Ainsi, l'utilisateur est seulement tenu de fournir la topologie du système en utilisant des modèles de la bibliothèque et les comportements des clients et des serveurs.

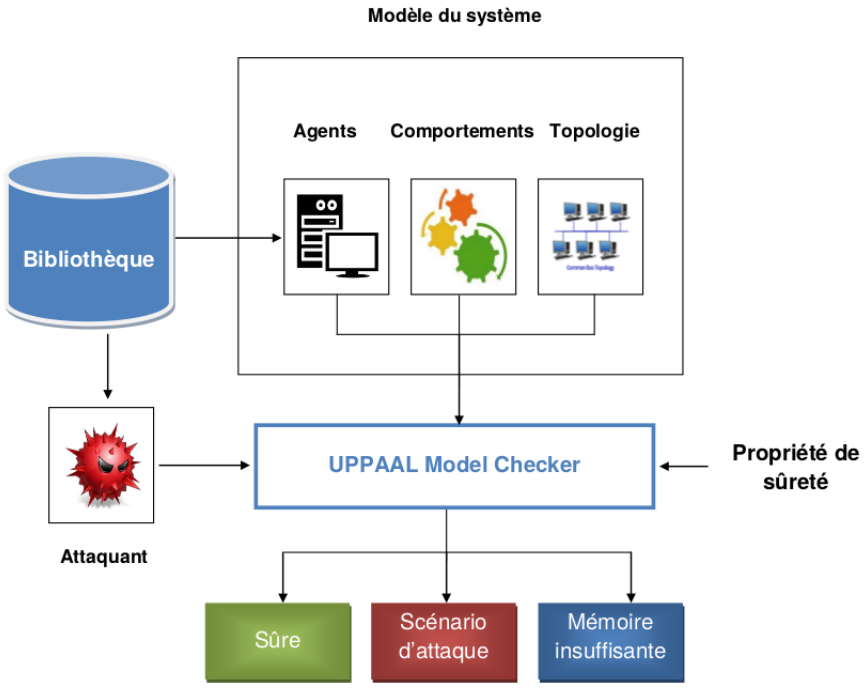


FIGURE 10 – Architecture du framework.

4.2 Le modèle du système

Dans UPPAAL, nous modélisons les agents qui interagissent avec les attaquants par la composition des automates temporisé comme présenter dans la Figure 11. Les clients peuvent créer, envoyer les demandes et recevoir des réponses pendant que le serveur peut recevoir les demandes, envoyer les réponses et exécuter des actions en fonction des demandes des clients. Les attaquants agissent comme l'attaquant Man-In-The-Middle et ont des capacités différentes en fonction de la configuration. Parmi ces capacités, ils peuvent écouter le réseau, intercepter, forger, rejouer ou modifier certains messages selon leurs connaissances. Un attaquant avec toutes ces capacités et un système de déduction s'appelle Dolev-Yao [6].

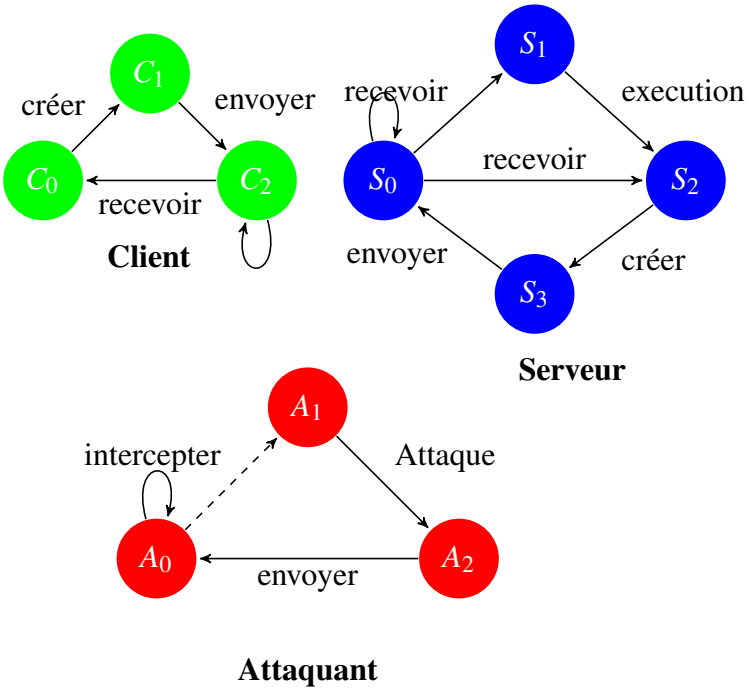


FIGURE 11 – Le diagramme de transition d'état du système.

Dans notre framework nous avons modéliser six automates nommés : *Client*, *BehaviorClient*, *Server*, *BeaviorServer*, *SecureDATA* et *Attacker*. Ils accèdent à des variables globales telles que les clés cryptographiques, les messages échangés sur les canaux, ainsi que les variables système. Les messages sont formatés à l'aide de la structure de données $\langle fonction, variable, valeur \rangle$ où $function$ est un variable booléenne qui exprime le type de fonction (lecture ou écriture), $variable$ est un entier qui désigne les différentes variables du système, et $valeur$ est un booléenne qui représente la valeur de la variable.

4.2.1 Les Modèles de clients

La bibliothèque de notre framework comporte plusieurs modèles de clients : un client MODBUS, un client OPC-UA (mode None, mode Sign et mode Sign Encrypt) et un client multi protocole qui peut communiquer avec dse serveurs MODBUS et OPC-UA. La figure 12 montre les interactions entre les automates : Client, BehaviorClient, SecureData et Server.

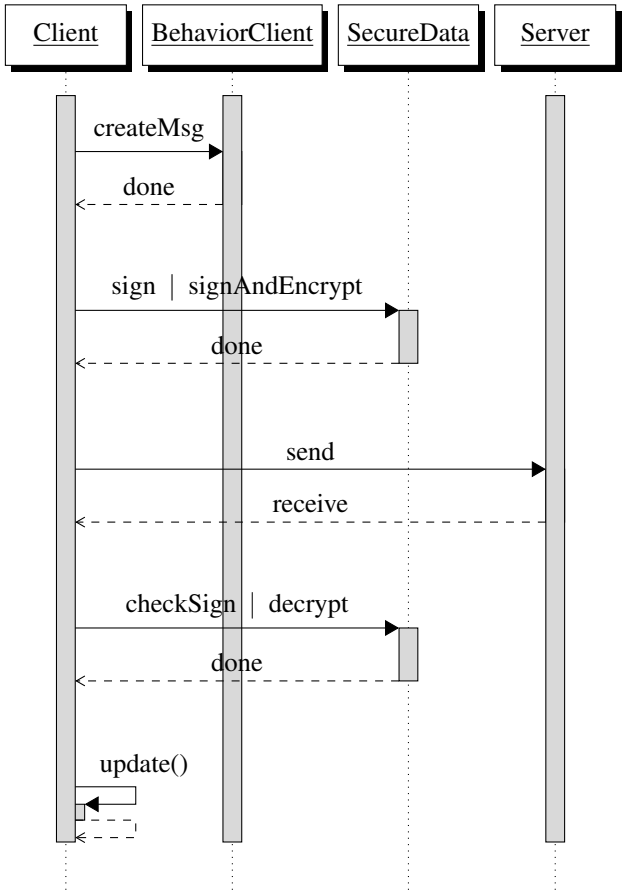


FIGURE 12 – Diagramme de séquence du client

Pour envoyer un message, l'automate *Client* commence par l'envoi de la première demande au *BehaviorClient* pour obtenir le contenu applicatif. Ensuite, dans le cas du client OPC-UA avec le mode sécurisé, le client sécurise (signe et / ou crypte) le message à l'aide de l'automate *SecureData*.

4.2.2 Les Modèles de Serveurs

Aussi pour les serveurs la bibliothèque du framework comporte plusieurs modèles de serveurs : un serveur MODBUS, un serveur OPC-UA (mode None, mode Sign et mode Sign En-

crypt). La figure 13 montre les interactions entre les automates : Server, BehaviorServer, SecureData et Client.

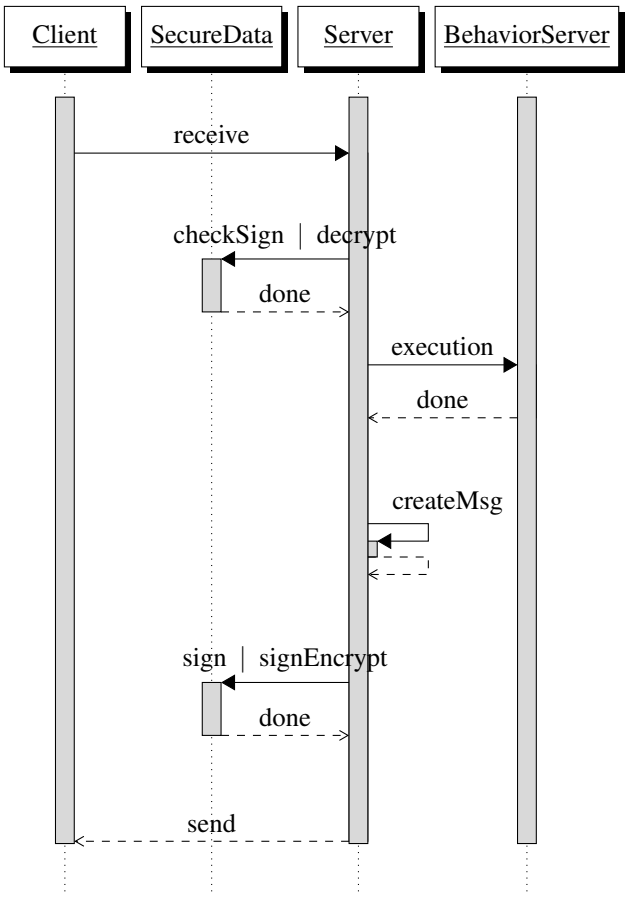


FIGURE 13 – Diagramme de séquence du serveur

En ce qui concerne l'automate du serveur, il attend un message envoyé par le client. Lorsque ce message est reçu, si le serveur implémente un protocole sécurisé (par exemple OPC-UA en mode sécurisé), il vérifie la signature du message. Ensuite, selon le type de message (lecture / écriture), Il modifie la valeur de la variable adressée ou lit sa valeur actuelle. Dans tous les cas, le serveur crée et envoie une réponse (éventuellement chiffrée) au client.

4.2.3 Le modèle SecureData

Les primitives cryptographiques sont représentées par un automate SecureData, qui permet de chiffrer, déchiffrer, signer ou vérifier la signature. La figure 14, présente l'automate SecureData qui a été implémenté dans le model checker UPPAAl. Pour chiffrer un message, on doit calculer $m+2p$, m est le message et p représente la clé publique. Le déchiffrement s'effectue avec $m-s$, s représente la clé privée dont $s = 2p$. Les agents de réseaux peuvent utiliser cet automate à l'aide des canaux de communication sign, signAndCrypt, decrypt et checkSign.

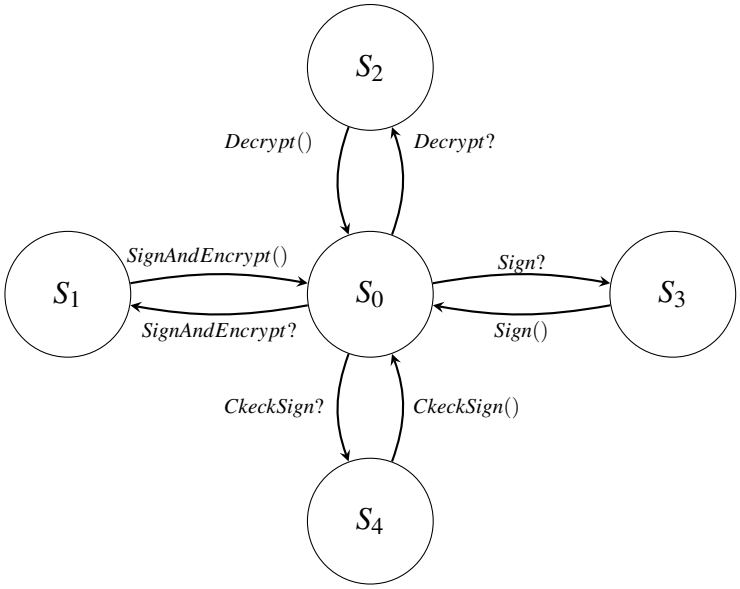


FIGURE 14 – Automate SecureData.

4.3 Modèles d'attaquants

Nous considérons quatre attaquants avec des capacités différentes, chacun étant modélisé comme un automate. En raison de la complexité des attaquants A_1, A_2 et A_3, nous devons limiter le nombre d'actions qu'ils peuvent effectuer lors d'une attaque en utilisant la variable i.

4.3.1 Attaquant 1

La figure 15 représente un attaquant qui fonctionne comme *Dolev-Yao* [6]. Il peut recevoir et envoyer les messages à travers le canal de communication entre le client et le serveur. Si l'attaquant déplace à l'état A_1, il peut décider de faire les actions :

- sélectionner une clé publique.
- sélectionner une clé privée.
- copier un message.
- rejouer un message.
- forger un message.
- chiffrer, signer et chiffrer ou déchiffrer un message.
- modifier une partie de message.

De plus, l'attaquant peut continuer à faire ces actions, choisir d'envoyer un message ou de bloquer un message.

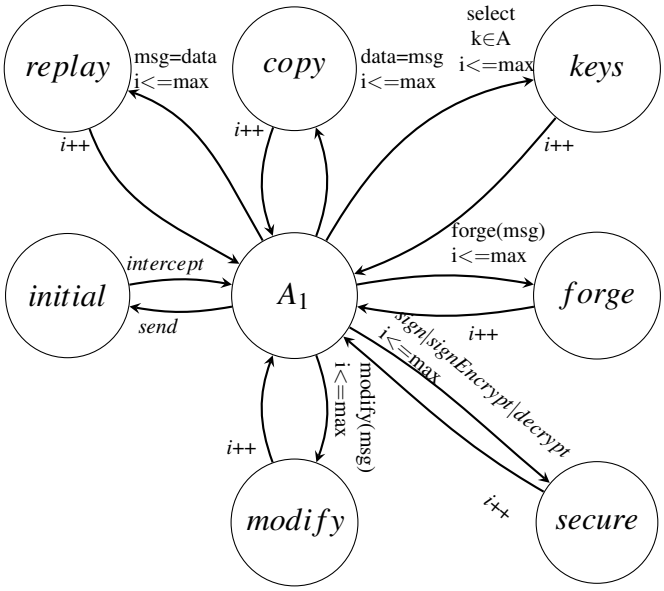


FIGURE 15 – Automate de l'attaquant A_1

4.3.2 Attaquant 2

Le rôle principal de cet attaquant est de modifier une partie du message ou le message complet à partir de ses connaissances. Si l'attaquant déplace à l'état A_2, il peut décider de faire les actions :

- sélectionner une clé publique.
- sélectionner une clé privée.
- chiffrer, signer et chiffrer ou déchiffrer un message.
- modifier une partie de message.

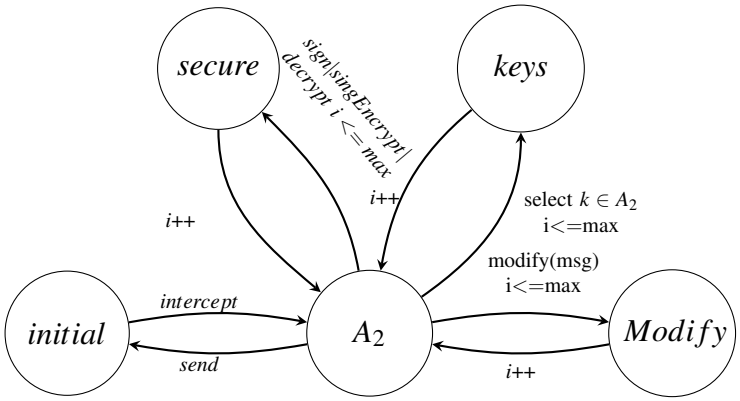


FIGURE 16 – Automate de l'attaquant A_2

4.3.3 Attaquant 3

Le rôle principal de cet attaquant est de forger les messages par la technique force brute à partir de ses connaissances. Si l'attaquant déplace à l'état A_3, il peut décider de faire les actions :

— sélectionner une clé publique.
— sélectionner une clé privée.
— chiffrer, signer et chiffrer ou déchiffrer un message.
— forger un message message.

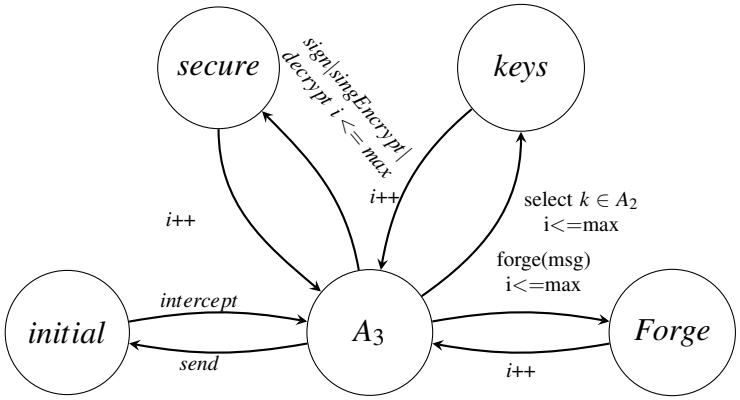


FIGURE 17 – Automate de l'attaquant A_3

4.3.4 Attaquant 4

Le rôle principal de cet attaquant est de rejouer les messages. Pour rejouer les messages l'attaquant d'abord se deplace vers l'état *copier* pour copier le message dans la variable *data* et après se deplacer vers l'état *rejouer* pour rejouer le message. La figure ci-dessous, représente l'automate de l'attaquant 4 :

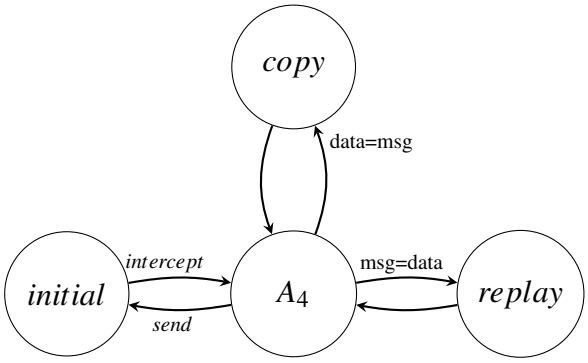


FIGURE 18 – Automate d'attaquant A_4

4.4 La spécification des propriétés

Dans notre framework on a verifié les propriétés de sûreté (Safety). Une propriété de sûreté permet de spécifier que ***quelque chose de mauvais n'arrivera jamais***. Ainsi, une propriété de sûreté peut seulement être satisfaite en un temps infini (on n'est jamais sûre) mais violée en un temps fini (quand le mauvais arrive) [9].

Exemple. les propriétés suivantes sont des propriétés de sûreté :
- absence de blocage.
- absence de l'exclusion mutuelle.
- x est toujours différent de 0.

Formalisation. une propriété de sûreté peut être formulée avec la logique temporelle sous la forme $A\Box\neg(\phi)$, où ϕ peut être un variable, nom d'état,...

4.5 Bibliothèque

Notre framework intègre une bibliothèque de composants comme : les attaquants, les agents et les topologies sous format XML.

5 Étude de cas

Dans cette section, nous illustrons notre approche avec un exemple d'étude de cas. Nous montrons comment nous l'avons implémenté dans le model checker UPPAAL et nous discutons les résultats obtenus en composant les différents modèles d'attaquant et topologies.

5.1 Description de l'étude de cas

Nous avons appliqué notre approche sur un exemple d'une usine de remplissage de bouteilles qui provient du simulateur de procédé VirtualPlant et est présenté en Figure 19. Des bouteilles vides avancent sur un tapis roulant. Un capteur détecte lorsqu'elles sont en position et actionne une valve faisant couler le liquide. Un autre capteur permet de détecter lorsque la bouteille est pleine. La valve se referme et le tapis roulant redémarre. Enfin, un client peut démarrer et arrêter tout le processus.

Dans cet exemple le comportement du procédé se compose de cinq variables booléennes :

$$V_p = \{\, motor,\ nozzle,\ levelHit,\ bottleInPlace,\ processRun \}$$

Ils désignent respectivement *motor* le tapis roulant, *nozzle* la valve, *levelHit* le capteur de niveau de liquide, *bottleInPlace* le capteur de position et *processRun* pour démarrer ou arrêter le procédé. La figure 20 montre un automate décrivant le comportement du procédé. Trois états sont

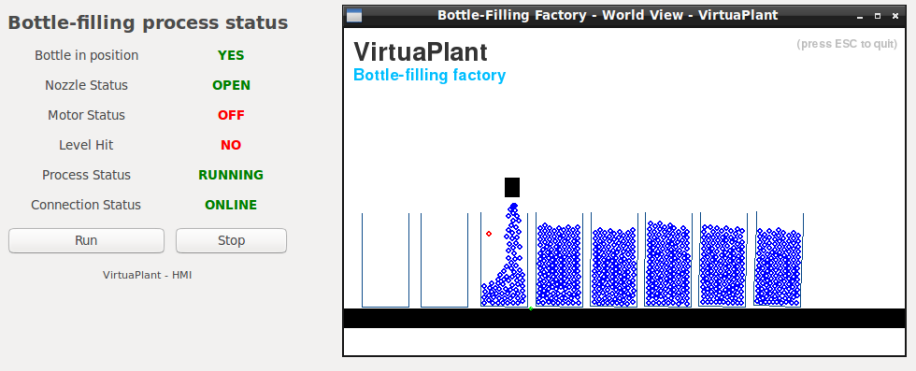


FIGURE 19 – Simulateur de VirtualPlant.

pris en compte : *Idle* signifie que le processus est arrêté, *Moving* que le tapis roulant fonctionne et les bouteilles avancent vers la valve, *Pouring* que état valve remplit une bouteille.

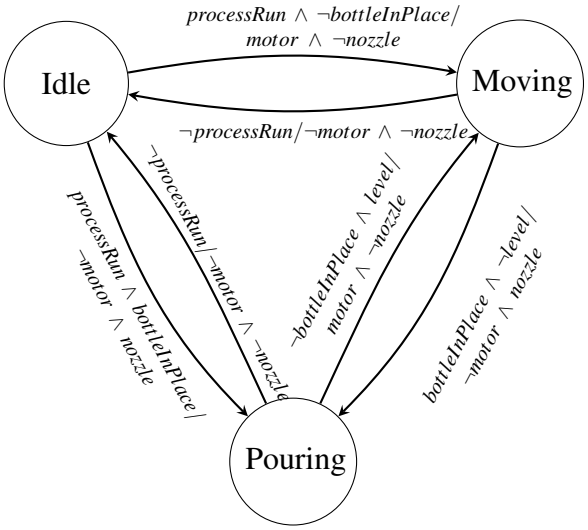


FIGURE 20 – Comportement du processus

Le comportement du client Selon l'automate présenté en Figure 21, le client ne démarrera et n'arrêtera tout le processus que lorsqu'il veut.

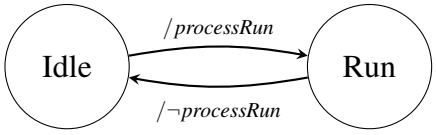


FIGURE 21 – Comportement du client

5.2 Spécification des propriété de sûreté

Les propriétés de sûreté que nous voulons que le processus garantis se sont les suivantes :

ϕ_1 La valve ne s'ouvre que lorsqu'une bouteille est en position.
$A\Box\neg(nozzle = true\ and\ bootle = false)$

ϕ_2 Le moteur ne démarre que lorsqu'une bouteille est pleine.
$A\Box\neg(motor = true\ and\ level = false)$

ϕ_3 La valve ne s'ouvre que lorsque le moteur s'arrête.
$A\Box\neg(nozzle = true\ and\ motor = true)$

5.3 Topologies

Nous considérons deux topologies. Dans la première topologie on n'utilise que le protocole MODBUS. Ce protocole est parmi les plus utilisés dans les communications industrielles et ne fournit aucun mécanisme de sécurité. La deuxième topologie que nous considérons divise le processus en deux serveurs avec deux protocoles MODBUS et OPC-UA. Nous supposons que le mode de sécurité SignEncrypt est utilisé dans le protocole OPC-UA.

Dans la $Topologie_1$ présentée dans la figure 22, un seul serveur MODBUS contrôle l'ensemble du procédé. Il communique avec un client MODBUS qui contrôle l'arrêt et le démarrage du procédé et lit l'état des capteurs, du tapis roulant et de la valve.

— Ensemble des serveurs $S = \{S_{MODBUS}\}$ avec :
- variables $V_{S_{MODBUS}} = V_p$
— Ensemble des clients $C = \{C_{MODBUS}\}$ avec :
- variables $V_{C_{MODBUS}} = \{\ processRun\ \}$

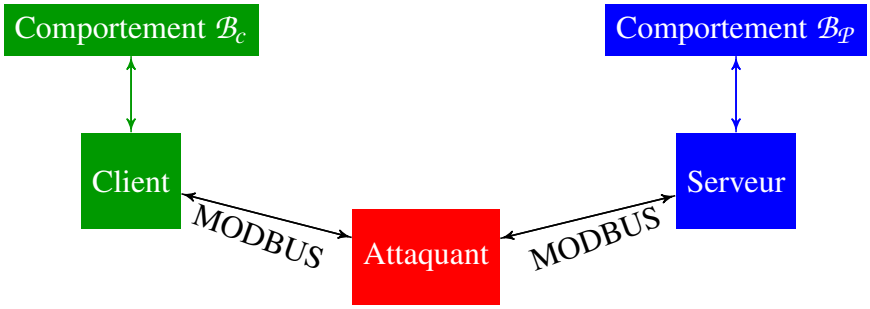


FIGURE 22 – $Topologie_1$

Dans la $Topologie_2$ présentée dans la figure 23, le tapis roulant et son capteur avec le démarrage et l'arrêt du procédé sont contrôlés par un serveur MODBUS et la valve et son capteur par un serveur OPC-UA (configuré en mode SignEncrypt). Un unique client multi-protocoles communique avec les deux serveurs.

— Ensemble des serveurs $S = \{S_{MODBUS}, S_{OPC-UA}\}$ avec :
- variables $V_{S_{MODBUS}} = \{ processRun, motor, bottleInPlace \}$
- variables $V_{S_{OPC-UA}} = \{ nozzle, levelHit \}$

— Ensemble des clients $C = \{C_{MODBUS}\}$ avec :
- variables $V_{C_{MODBUS}} = \{ processRun \}$

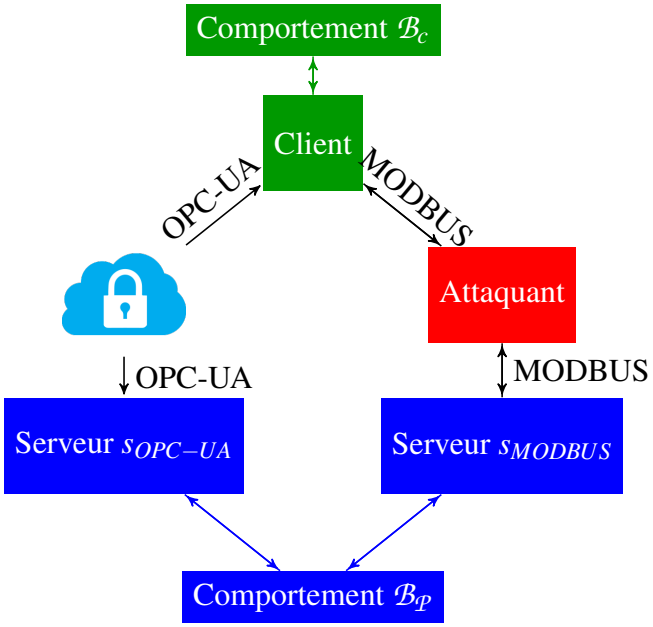


FIGURE 23 – $Topologie_2$

5.4 Attaquants

Pour montrer la modularité de notre framework, nous testons les deux topologies contre les quatre attaquants proposés dans la section 4.3. Pour rappel, le premier attaquant présenté dans la Figure 15 est basé sur le modèle Dolev-Yao [6], il peut écouter le réseau, intercepter, forger, rejouer ou modifier certains messages selon ses connaissances. Le deuxième attaquant présenté à la Figure 16 ne peut modifier que les messages. Le troisième attaquant présenté à la Figure 17 ne peut que forger de nouveaux messages. Enfin, le dernier attaquant présenté à la Figure 18 ne peut reproduire que les messages envoyés par le client et le serveur.

5.5 Résultats obtenus avec UPPAAL

Aprés avoir expérimenté les différents algorithmes dans UPPAAL, nous avons choisi d'appliquer l'algorithme *Breadth first search* et *Compact Data Structure* pour représenter l'espace d'états. Les résultats sont résumés dans le Tableau 1 où ✓ veut dire qu'une attaque a été trouvée et ✗ signifie que la propriété est satisfaite, aussi ◆ veut dire que UPPAALne pouvait pas conclure. Cela s'est produit parce que l'outil demande plus de mémoire que disponible. Nos expériences ont été menées sur un Intel(R) Core(TM) i5-4590 CPU @ 3.30GHz avec 16GB de RAM.

		$Attaquant_1$	$Attaquant_2$	$Attaquant_3$	$Attaquant_4$
$Topologie_1$	Φ_1	✓	✓	✓	✗
	Φ_2	✓	✓	✓	✗
	Φ_3	✓	✓	✓	✗
$Topologie_2$	Φ_1	◆	◆	✗	✗
	Φ_2	✓	✓	✓	✗
	Φ_3	✓	✓	✓	✗

TABLE 1 – Résultats obtenus

En théorie, aucun des quatre attaquants ne peut violer la propriété Φ_1 dans $Topologie_2$ puisque le serveur OPC-UA contrôle la variable *nozzle*. Même avec le serveur MODBUS qui contrôle la variable *bottleInPlace*, si *bottleInPlace* est forcé à *Faux* par un attaquant alors que la variable *nozzle* est *vraie*. Ensuite *nozzle* passe automatiquement en *faux* en raison du comportement du processus (et vice versa). Ainsi, la seule façon de le faire est de forcer l'ouverture du *nozzle* qui n'est pas possible dans $Topologie_2$ (comme on peut le voir avec les attaquants A_3 et A_4). De même, l'attaquant A_4 ne peut pas violer les propriétés, puisque les messages transmisent entre le client et le serveur ne sont que pour démarrer ou arrêter le processus.

		$Attaquant_1$	$Attaquant_2$	$Attaquant_3$	$Attaquant_4$
$Topologie_1$	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
$Topologie_2$	Φ_1	Mémoire insuffisante		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 2 – Temps de vérification pour chaque propriété

Selon les tableaux 1 et 2, l'attaquant A_2 obtient les mêmes résultats que A_1 (Dolev-Yao) dans un temps plus court. L'attaquant A_3 prend un peu plus de temps mais peut conclure sur la propriété Φ_1 dans la $Topologie_2$ alors que les attaquants A_1 et A_2 ne peuvent pas trouver de résultats. Cela montre que les attaquants puissants tels que Dolev-Yao sont souvent trop complexes et seulement certaines parties de l'attaquant suffisent à trouver des attaques. Ils sont cependant critiques lorsqu'ils tentent de prouver l'absence d'attaques. D'autre part, l'attaquant A_4 obtient des temps plus importants qui peuvent être surprenants car c'est le plus simple de nos attaquants. Une explication probable est que, puisque tous ces résultats sont l'absence d'attaques, UPPAAL doit explorer tous les états possibles ce qui peut prendre plus de temps que trouver un contre-exemple.

La Figure 24 montre un scénario d'attaque avec A_2 contre Φ_2 dans $Topology_2$. Le client envoie un message au serveur MODBUS pour démarrer le processus, le moteur démarre et les bouteilles avancent sur le tapis roulant. Après un certain temps, le client envoie un message pour arrêter le processus. L'attaquant intercepte le message et modifie la variable ciblée par la demande d'écriture et la nouvelle valeur pour forcer le démarrage du moteur.

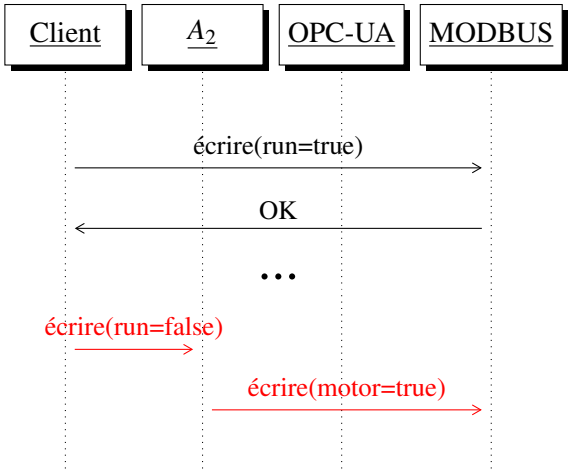


FIGURE 24 – Scénarion d'attaque avec A_2 contre Φ_2 dans $Topology_2$

Cette expérimentation montre que nous n'avons pas besoin de toute la puissance de Dolev-Yao pour trouver des attaques. Ainsi il aide également à trouver quelles sont les actions nécessaires pour que l'attaquant puisse mener une attaque donnés.

5.6 Limite de notre approche

Notre approche est basée sur le model checker et la principale limite à l'utilisation du model checker est liée au problème de l'*explosion combinatoire*, le nombre d'états augmente de façon exponentielle en fonction de la complexité du système. On peut voir la limite dans les Tables 1 et 2 où le model checker ne peut pas trouver des résultats sur la propriété Φ_1 dans la $Topologie_2$ avec les attaquants : $Attaquant_1$ et $Attaquant_2$.

6 Conclusion

Nous avons développé une approche modulaire pour évaluer la sécurité des systèmes de contrôle industriel SCADA. Cette approche vise à trouver des attaques applicatives en tenant compte de différents paramètres tels que le comportement du processus, les propriétés de sûreté, ainsi que les positions et les capacités des attaquants. Nous avons montré comment cette approche peut être implémentée à l'aide du modèle-checker UPPAAL. Nous l'avons appliqué sur un exemple et on a montré comment la variation des propriétés, des topologies de réseau et des attaquants peut modifier les résultats obtenus.
Dans le futur proche, nous sommes intéressés à étudier plusieurs processus (éventuellement imbriqués) qui nécessiteraient une représentation d'automates plus complexe. De plus, nous cherchons à généraliser la mise en œuvre UPPAAL et à créer un outil open source pour générer automatiquement des modèles UPPAAL et interpréter les résultats.

Références

[1] Stuart A.BOYER : *Scada : Supervisory Control And Data Acquisition*. International Society of Automation, USA, 4th édition, 2009. :Boyer-2009

[2] Christel BAIER et Katoen JOOST-PIETER : *Principles of Model Checking*. The MIT Press, 2008. :BC-KJP2008

[3] Gerd BEHRMANN, Re DAVID et Kim G. LARSEN : A tutorial on uppaal. pages 200–236. Springer, 2004. :Uppaal-Behrmann04atutorial

[4] Johan BENGTSSON et Wang YI : Timed automata : Semantics, algorithms and tools. *In* Jörg DESEL, Wolfgang REISIG et Grzegorz ROZENBERG, éditeurs : *Lectures on Concurrency and Petri Nets*, volume 3098, pages 87–124. Springer, 2003. :Bengtsson-2003

[5] Edmund CLARKE, Orna GRUMBERG et Doron PELED : *Model Checking*. MIT Press, 1990. :EC90

[6] D. DOLEV et A. YAO : On the security of public key protocols. *In Proceedings of the 22Nd Annual Symposium on Foundations of Computer Science*, pages 350–357, Washington, DC, USA, 1981. IEEE Computer Society. :Dolev-1981

[7] Jannik DREIER, Maxime PUYS, Marie-Laure POTET, Pascal LAFOURCADE et Jean-Louis ROCH : Formally verifying flow integrity properties in industrial systems. *In In SECRYPT 2017 - 14th International Conference on Security and Cryptography*, 2017. :JD-2017

[8] Mathias EKSTEDT, Pontus JOHNSON, Robert LAGERSTRÖM, Dan GORTON, Joakim NYDRÉN et Khurram SHAHZAD. : Securi cad by foreseeti : A cad tool for enterprise cyber security management. *In In Enterprise Distributed Object Computing Workshop (EDOCW), 2015 IEEE 19th International*, page 152–155. IEEE, 2015. :ME

[9] Marion GUTHMULLER : *Dynamic formal verification of temporal properties on legacy distributed applications*. Theses, Université de Lorraine, 2015. :guthmuller-2015

[10] Hannes HOLM, Teodor SOMMESTAD, Mathias EKSTEDT et Lars NORDSTROM : Cysemol : A tool for cyber security analysis of enterprises. *In In Electricity Distribution (CIRED 2013), 22nd International Conference and Exhibition on. IET*, page 1–4, 2013. :HH

[11] Peter HUITSING, Rodrigo CHANDIA, Mauricio PAPA et Sujeet SHENOI : Attack taxonomies for the modbus protocols. *IJCIP*, 1:37–44, 2008. :H-2008

[12] IEC-62443 : Industrial communication networks - network and system security. *In International Electrotechnical Commission*, 2010. :IEC-2010

[13] Sung-Hwan KIM, Jung-Ho EOM et Tai-Myoung CHUNG : A study on optimization of security function for reducing vulnerabilities in scada. *In CyberSec*, pages 65–69. IEEE, 2012. :Kim-2012

[14] Ronald L KRUTZ : *Securing SCADA Systems*. Wiley, 2005. :krutz-2005

[15] Ralph LANGNER : Stuxnet : Dissecting a cyberwarfare weapon. *IEEE Security and Privacy*, 9:49–51, 2011. :LR-2011

[16] Robert M. LEE, Michael J. ASSANTE et Tim CONWAY : German steel mill cyber attack. *Industrial Control Systems 30*, 2014. :RM-2014

[17] Robert M. LEE, Michael J. ASSANTE et Tim CONWAY : Sans industrial control systems. *Analysis of the cyber attack on the Ukrainian power grid*, 2016. :RM-2016

[18] Wolfgang MAHNKE, Stefan-Helmut LEITNER et Matthias DAMM : *OPC Unified Architecture*. Springer Publishing Company, Incorporated, 1st édition, 2009. :Mahnke-2009

[19] Xinming OU, Sudhakar GOVINDAVAJHALA et Andrew W APPEL : Mulval : A logic-based network security analyzer. *In In USENIX security*, 2005. :XO

[20] John PARK, Steve MACKAY et Edwin WRIGHT : *Practical Data Communications for Instrumentation and Control*. Elsevier Science, 2003. :M-2003

[21] Maxime PUYS, Marie-Laure POTET et Abdelaziz KHALED : Generation of applicative attacks scenarios against industrial systems. *In International Workshop on Interplay of Security, Safety and System/Software Architecture (ISSA)*. ACM ICPS, 2017. :Abdelaziz-2017

[22] Maxime PUYS, Marie-Laure POTET et Pascal LAFOURCADE : Formally verifying flow integrity properties in industrial systems. *In 35th International Conference, SAFECOMP*, page 67–75, 2016. :MP-2017

[23] Robert RADVANOVSKY et Jacob BRODSKY : *Handbook of SCADA/Control Systems Security, Second Edition*. CRC Press, Inc., Boca Raton, FL, USA, 2nd édition, 2016. :Radvanovsky-2016

[24] S.KRIAA, M.BOUISSOU et Y.LAAROUCHI : A model based approach for scada safety and security joint modelling : S-cube. *In In IET System Safety and Cyber Security. IET Digital Library*, 2015. :SK-2015

[25] Pallapa VENKATARAM, Sunilkumar S. MANVI et B.Babu SATHISH : *Communication Protocol Engineering*. Prentice-Hall of India Pvt.Ltd, 2 édition, 2014. :Venkataram-2014