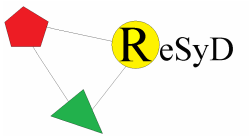
République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Abderrahmane Mira de Bejaïa

Faculté des Sciences Exactes
Département d'Informatique
Ecole Doctorale Réseaux et Systèmes Distribués

Mémoire de Magistère
En Informatique

Option
Réseaux et Systèmes Distribués



Thème

An Integrated Approach for Protecting Data In SCADA Systems (AI3S)

Présenté par
LABENI Merouane

Devant le jury composé de :

Président :	**KERKAR Moussa**	**Professeur**	**Université A/Mira, Bejaia, Algérie**
Rapporteur :	**TARI Zahir**	**Professeur**	**Université RMIT, Melbourne, Australie**
Examinateurs :	**MELIT Ali**	**M.C.A**	**Université de Jijel, Algérie**
	BOUKERRAM Abdellah	**M.C.A**	**Université de Sétif, Algérie**
Invité :	**TARI Abdelkamel**	**M.C.B**	**Université A/Mira, Bejaia, Algérie**

Promotion 2008-2009

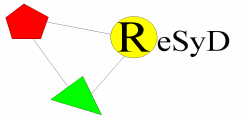
République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Abderrahmane Mira de Bejaïa

Faculté des Sciences Exactes
Département d'Informatique
Ecole Doctorale Réseaux et Systèmes Distribués

Mémoire de Magistère
En Informatique

Option
Réseaux et Systèmes Distribués



Thème

Une Approche Intégrée pour la Protection des Données dans les Systèmes SCADA (AI3S)

Présenté par
LABENI Merouane

Devant le jury composé de :

Président :	**KERKAR Moussa**	**Professeur**	**Université A/Mira, Bejaia, Algérie**
Rapporteur :	**TARI Zahir**	**Professeur**	**Université RMIT, Melbourne, Australie**
Examinateurs :	**MELIT Ali**	**M.C.A**	**Université de Jijel, Algérie**
	BOUKERRAM Abdellah	**M.C.A**	**Université de Sétif, Algérie**
Invité :	**TARI Abdelkamel**	**M.C.B**	**Université A/Mira, Bejaia, Algérie**

Promotion 2008-2009

Dédicaces

A ma mère
A toute ma famille

Remerciements

Je tiens à exprimer mes reconnaissances aux personnes qui ont contribué de près ou de loin pour accomplir ce travail. Cet espace ne suffira sûrement pas pour citer toutes ces personnes. Qu'elles trouvent ici mes sincères remerciements.

Mes vifs remerciements accompagnés de gratitude sont adressés à mon encadreur M^{r} TARI Zahir, Professeur à l'université RMIT de Melbourne, Australie, pour avoir dirigé ce mémoire et pour ses conseils précieux.

Je tiens à remercier cordialement M^{r} TARI Abdelkamel, Docteur et responsable de l'école doctorale en informatique à l'université de Béjaïa de m'avoir co-encadré, orienté et encouragé.

Mes remerciements vont aussi aux membres de jury pour avoir voulu juger mon travail et l'intérêt qu'ils y portent. J'adresse mes remerciements à Monsieur KERKAR Moussa d'avoir présidé le jury. Je tiens à remercier Messieurs MELIT Ali et BOUKERRAM Abdellah pour avoir accepté de faire partie de jury et d'avoir examiné ce travail.

Je ne pourrai oublier d'adresser ma reconnaissance, mes remerciements et ma profonde gratitude à ma famille qui est toujours derrière ma réussite. Je remercier tous mes amis qui m'ont accompagné le long de tout mon cursus et tous mes collègues de l'école doctorale ReSyD de Béjaia.

Table des Matières

Liste des Figures

Liste des Tableaux

Liste des Algorithmes

Introduction Générale

Les systèmes d'ordinateurs basés sur Le contrôle et l'acquisition de données de surveillance (SCADA) ont évolué au cours des 40 dernières années, d'autonomie, des opérations compartimentées vers les architectures distribuées qui communiquent à travers des grandes distances. En outre, leurs implémentations ont émigré du matériel et du logiciel faits sur mesure aux plateformes de matériel et de logiciel standard. Ces changements ont mené à la réduction des coûts de développement, d'exploitation, et d'entretien aussi bien que fournir la gestion exécutive à l'information temps-réel qui peut être employée pour soutenir la planification, la surveillance, et la prise de décision. Ces avantages, cependant, viennent avec un coût. Les systèmes de contrôle industriels semi-isolés qui utilisent le matériel et le logiciel propriétaire[1] sont maintenant vulnérables aux intrusions à travers les réseaux externes, y compris l'Internet, aussi bien que du personnel interne. Ces attaques tirent profit des vulnérabilités dans les plateformes standard, telles que Windows, et des PCs qui ont été adoptés pour l'usage dans les systèmes SCADA.

Cette situation pourrait être considérée une progression normale de souci modéré (comme dans beaucoup d'autres secteurs en utilisant les systèmes numériques) si elle n'était pas pour le fait que les systèmes SCADA contrôlent un grand pourcentage des infrastructures critiques d'Europe et du monde, telles que les centrales nucléaires, les installations de production de l'électricité, les canalisations (Gaze, Pétrole, etc.), les

[1] De propriété industrielle (trademark)

raffineries et les usines chimiques. En outre, elles sont directement et indirectement impliquées à la fourniture des services aux ports maritimes, aux systèmes de transport, aux canalisations, aux usines, et à beaucoup d'autres entreprises critiques.

La tolérance d'intrusion est une nouvelle approche qui a lentement émergé pendant la dernière décennie, et gagne un essor lumineux récemment, elle est le déclencheur principal de l'évolution du secteur de la fiabilité. Dans ce contexte, la tolérance de fautes aura l'applicabilité dans tous les domaines même ceux dites critiques. En outre, un mécanisme qui vérifie les flux d'information lors des échanges de données entre les objets d'un système est indispensable pour augmenté sa résilience conter les tentatives malveillantes. Le contrôle de flux d'information assure la confidentialité des informations même après leur dissémination.

Notre approche consiste en un modèle hybride intégrant la tolérance d'intrusion et le contrôle de flux d'information afin d'obtenir une technique plus résiliente et plus puissante en terme de sécurité, et elle est globale puisqu'elle est définie d'une manière formelle et permet de spécifier les concepts de tolérance d'intrusion –et leur importance pour établir la résilience des systèmes repliés– et des règles de flux d'informations. Ainsi le modèle proposé contribue au renforcement de la protection des systèmes d'infrastructure critiques ICs (i.e. SCADA, PCS, etc.).

Notre objectif est de proposer un modèle hybride pour sécuriser les systèmes critiques en se basant sur les points suivants : i) étude et vérification des deux modèles, de contrôle de flux d'information proposé par C. Bryce, J.P. Banâtre, D. Le Métayer [*BRY95*] et de tolérance d'intrusion pour les systèmes SCADAs proposé par P. Sousa, A. N. Bessani, M. Correia, N. F. Neves et P. Verissimo [*SOU07*]; ii) amélioration du modèle de Contrôle de Flux d'Information en définissant de nouvelles règles et sémantiques pour assurer non seulement la confidentialité des informations mais aussi leur intégrité; iii) intégration des deux modèles précédents afin d'augmenter la résilience des systèmes à base d'information critiques en corrigeant certains points de vulnérabilité touchant ces systèmes.

Afin de présenter une vue complète des concepts de sécurité des systèmes SCADA et de leur rôle important dans les infrastructures critiques (ICs), ce mémoire commence en définissant les composants et les fonctions d'un système SCADA, et en fournissant des illustrations des architectures générales de tels systèmes. Avec ce contexte, des réalisations spécifiques de SCADA dans une variété d'applications

critiques sont présentées avec une détermination des soucis de sécurité et des issues nocifs potentiels des attaques sur ces opérations. Le deuxième chapitre introduit la tolérance d'intrusion (TI) d'une manière générale et nous présentons des concepts et quelques mécanismes de tolérance d'intrusion puis nous présentons un modèle type pour la protection des infrastructures d'informations critiques. Le chapitre suivant, quant à lui, introduit le mécanisme de contrôle de flux d'information et les travaux importants qui s'inscrivent dans ce contexte.

Nous présentons le modèle hybride proposé dans le quatrième chapitre où nous définissons clairement notre contribution pour l'amélioration du modèle de Bryce, Banâtre & Le Métayer [*BRY95*], et nous donnons une architecture logique distribuée pour le système de tolérance d'intrusion et de contrôle de flux d'information. Le cinquième chapitre est consacré au développement formel du modèle de sécurité proposé. Ce chapitre contient les définitions nécessaires des concepts de bases du modèle, la définition de nouvelles règles, des algorithmes et des démonstrations concernant la sureté des flux d'informations échangés entre les objets de système.

CHAPITRE

1

Les Systèmes SCADA et leur Sécurité

1.1 Introduction

Les systèmes de commande et d'acquisition de données de surveillance (SCADA : Supervisory Control And Data Acquisition) sont les composants essentiels des infrastructures critiques de la plupart des nations. Ils contrôlent des canalisations de gaz et de pétrole, des systèmes de l'eau et de transport, des installations, des raffineries, des usines chimiques, et une grande variété d'opérations de fabrication. SCADA fournit une gestion des données en temps réel sur des opérations de production, implémente des paradigmes de contrôle plus efficaces, améliore la sûreté de l'industrie et de personnel, et réduit les coûts d'opération. Ces avantages sont rendus possibles par l'utilisation des standards (matériels et logiciels) dans les systèmes SCADA combinés avec des protocoles de transmission améliorés et l'accroissement de la connectivité aux réseaux extérieurs, y compris l'Internet. Cependant, ces avantages sont acquis au prix du renforcement de la vulnérabilité aux attaques ou aux actions incorrectes d'une variété de sources extérieures et intérieurs [*API04*].

Ce chapitre explore l'évolution des systèmes SCADA, leurs caractéristiques, fonctions, applications typiques, et des titres généraux.

1.2 Évolution de SCADA, définitions, et architecture de base

Les systèmes de contrôle et d'acquisition de données de surveillance SCADA signifie différentes choses aux différentes personnes, selon leurs milieux et perspectives. Par conséquent, il est important de passer en revue l'évolution de SCADA et de sa définition comme compris par des professionnels et des praticiens dans le domaine [*DAC03*].

1.2.1 Évolution de SCADA

La portée de SCADA a évolué de ses commencements dans les années 60. L'arrivée des mini-ordinateurs peu coûteux tels que Digital Equipment Corporation PDP-8 et PDP-11 a rendu la gestion par ordinateur des processus et des opérations de fabrication plus faisables. Les contrôleurs logiques programmables (PLCs) ont progressé simultanément. Ces derniers ont mis en application la logique traditionnelle *relay ladder*[1] pour contrôler les processus industriels. PLC a fait appel aux automaticiens traditionnels qui ont été accoutumés à la logique de programmation de relais et qui n'ont pas voulu apprendre des langages de programmation et des logiciels d'exploitation. Quand des micro-ordinateurs ont été développés, ils ont été programmés et empaquetés pour émuler les PLCs en leur fonction, leur programmation, et leurs opérations. En fait, la concurrence a développée entre les deux approches et continue à ce jour.

Initialement, les systèmes de contrôle ont été confinés à une industrie particulière. Les dispositifs de contrôle associés étaient locaux à l'usine et non reliés à aucun réseau externe. Les systèmes de contrôle anciens se sont composés d'un mini-ordinateur central ou d'un PLC qui ont communiqué avec les contrôleurs locaux, se sont connectés par interface aux moteurs, pompes, valves, commutateurs, sondes, et ainsi de suite. La figure **1.1** illustre cette architecture. Cette architecture désignée parfois sous le nom d'un système *réparti de contrôle*. De tels systèmes sont généralement confinés aux endroits près de l'un l'autre, emploient normalement un réseau local à grande vitesse, et comportent habituellement le contrôle en circuit fermé. Comme condition nécessaire pour l'opération de ces systèmes, les compagnies

[1] Dispositif pour retransmettre un signal et l'amplifier.

et les fournisseurs ont développé leurs propres protocoles de communication, beaucoup dont étaient de propriété industrielle.

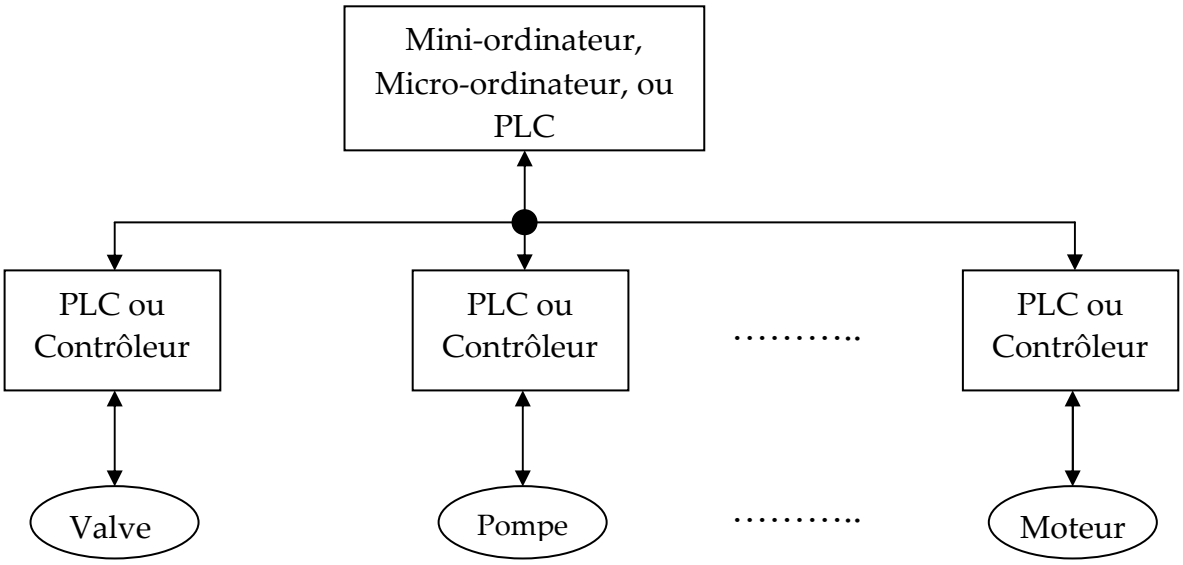


Figure 1.1: Système typique de contrôle local

Comme les possibilités techniques des ordinateurs, des systèmes d'exploitation, et des réseaux sont améliorés, la gestion d'organisation a poussé pour la plus grande connaissance du statut en temps-réel des opérations de l'industrie à distance. En outre, dans les organismes avec un certain nombre d'opérations géographiquement séparées, l'acquisition de données à distance, le contrôle, et l'entretien sont devenus de plus en plus attractifs des points de vue gestion et coût. Ces possibilités sont connues collectivement en tant que le contrôle et l'acquisition de données de surveillance ou SCADA.

1.2.2 Définitions

Nous listons ici deux définitions typiques d'un système SCADA et la source de chacune :

■■ SCADA est la technologie qui permet à un utilisateur de rassembler des données d'un ou plusieurs équipements éloignés et/ou d'envoyer des instructions limitées de contrôle à ces équipements. SCADA : le contrôle et l'acquisition de données de surveillance par Stuart A. Boyer, édité par ISA l'instrumentation, systèmes, et société d'automation ; 3ème édition.

■■ Un système fonctionnant avec des signaux codés sur des voies de communication afin de fournir le contrôle de l'équipement RTU (Remote Terminal Unit). IEEE C37.1-1994 standard, définitions, spécifications, et analyse des systèmes utilisés pour le contrôle de surveillance, l'acquisition de données, et le contrôle automatique. (Le RTU est discuté dans la prochaine section.)

Des définitions additionnelles liées aux systèmes SCADA sont données dans le tableau **1.1** [*I3E94*]. Cette liste n'est pas censée pour être inclusive, mais décrit quelques termes importants utilisés dans l'application des systèmes SCADA.

TERME	DÉFINITION
déterministe	Degré auquel une activité peut être exécutée dans un planning prévisible.
DeviceNet	Un protocole de réseau de contrôle d'Allen Bradley[2] qui est employé pour relier PLCs et contrôleurs locaux.
ControlNet	Un protocole de communication d'Allen Bradley s'est appliqué aux systèmes de contrôle.
Data Highway,	Les protocoles de communications d'Allen Bradley.
fieldbus	Protocoles de communication qui facilitent l'échange des messages entre les dispositifs du domaine. Quelques exemples des protocoles fieldbus sont Foundation Fieldbus, Modbus, DeviceNet, et Profibus.
hot stand-by system	Un système dupliqué qui est synchronisé avec le système principal et qui peut assurer le contrôle si le système principal tombe en panne.
proportional, integral, derivative(PID) control	méthode utilisée pour calculer des paramètres de contrôle pour maintenir un point de réglage prédéterminé. Des techniques mathématiques sont employées pour calculer les taux de changement, le temps de retard, et autre fonctions nécessaires pour déterminer les corrections à appliquer.
real-time (adjective)	Une action qui se produit au même rythme que le temps réel ; aucune latence, aucune durée du traitement.
real-time operating system (RTOS)	Un ordinateur du système d'exploitation qui implémente des processus et services d'une façon déterministe.

Tableau 1.1: Définitions liées au SCADA

[2] Fabricant de protocoles connu aussi sous le nom Rockwell

1.2.3 L'Architecture de système SCADA

Une terminologie spécifique est associée aux composantes des systèmes SCADA. Ces éléments sont définis comme suit [*CER01*]:

■■ Opérateur

S'agit de l'opérateur humain qui surveille le système SCADA et exécute les actions de contrôle et de surveillance pour les opérations d'industrie distante.

■■ Interface homme-machine (IHM)

Elle présente les données à l'opérateur et fournit pour le contrôle, des entrées (inputs) dans une variété de formats, y compris des graphiques, de schémas, fenêtres, menus déroulants, écrans à contact, et ainsi de suite.

■■ Unité terminale principale (MTU)

Le MTU est équivalent à une unité principale (maître) dans une architecture maître/esclave. Le MTU présente des données à l'opérateur à travers l'IHM, recueille des données de l'emplacement éloigné, et transmet des signaux de contrôle au site distant. Le taux de transmission de données entre le MTU et le site distant est relativement faible et la méthode de contrôle est habituellement boucle ouverte en raison du temps de retard ou les interruptions des flux de données.

■■ Moyens de communications

La méthode de communication entre le MTU et les contrôleurs à distance. La communication peut être à travers l'Internet, les réseaux sans fils ou câblés, ou le réseau téléphonique commuté.

■■ Unité terminal distante (RTU)

Elle fonctionne comme un esclave dans l'architecture maître/esclave. Envoie des signaux de commande au dispositif sous-contrôle, acquiert des données de ces dispositifs, et les transmet au MTU. Un RTU peut être un PLC[3]. Le débit entre le RTU et le dispositif commandé est relativement élevé et la méthode de contrôle est habituellement en circuit fermé.

[3] Programmable logic controller

Un diagramme général d'un système SCADA est montré sur la figure **1.2.** Les architectures modernes de SCADA se fondent fortement sur des protocoles standards et sur la transmission de données numérique. Par exemple, un protocole de transmissions tel que Foundation-Fieldbus, qui est mentionné dans le tableau ci-dessus, est appliqué en conjonction avec les radios industrielles d'Ethernet. Ces radios d'Ethernet fournissent des débits de 512 Kbps, une grande augmentation en plus de ceux fournis par des liens en séries EIA-232. Pour la sécurité, les points d'accès industriels d'Ethernet emploient la technologie du saut de fréquence (Frequency Hopping spread-spectrum)[4] avec le cryptage.
Comme discuté précédemment, une architecture SCADA comporte deux niveaux : un niveau maître ou client au centre de contrôle de surveillance et un niveau esclave ou serveur de données qui agit l'un sur l'autre avec les processus sous contrôle. En plus du matériel, les composants logiciels de l'architecture SCADA sont importants. Voici certains des composants logiciels typiques pour SCADA :

■■ SCADA maître/client
- – Interface homme-machine
- – Manipulation d'alarme
- – Événement et surveillance du journal (log monitoring)
- – Applications spéciales
- – ActiveX ou Java contrôles

■■ SCADA esclave/serveur de données
- – Gestionnaire du système en temps-réel
- – Applications de traitement de données
- – Générateur de rapport
- – Manipulation d'alarme
- – Pilotes et interfaces pour contrôler les équipements (composants)
- – Bilan (Spreadsheet)
- – Enregistrement de données
- – Archivage
- – Traçage et poursuite

[4] FHSS : elle consiste à découper la bonde en 75 canaux (sauts) et de transmettre, en utilisant une combinaison de canaux connue de toutes les stations de la cellule, successivement sur un canal puis sur un autre pendant une petite période, conçue dans un but militaire afin d'empêcher l'espionnage des informations.

Opérateur

Interface-Homme Machine IHM

Station de contrôle centrale (MASTER TERMINAL UNIT-MTU)

Communication via Internet, Wireless Network, Wired Network, ou Switched Public Telephone Network
(Débit relativement faible, contrôle habituellement en boucle ouvert)

Le domaine d'acquisition de données & unités de contrôle (REMOTE TERMINAL UNIT-RTU; may be Programmable Logic Controller-PLC)

(Débit relativement élevé, contrôle habituellement en boucle fermée)

Le domaine d'équipements de données comme Pompes, sondes, commutateurs, etc.

Figure 1.2: Architecture Typique d'un système SCADA

1.3 Les applications SCADA

SCADA est dominant dans le monde entier. Comme discuté précédemment, il imprègne les infrastructures critiques du monde, surveillant et commandant une variété de processus et d'opérations. Des exemples des systèmes courants SCADA sont montrés sur les figures **1.3** à **1.6** [*BYR00*] pour illustrer la diversité de leurs domaines d'application. Cependant, il est utile de noter les similitudes dans leurs architectures.

Dans certains exemples, les normes EIA-232 et EIA-485 sont employés. EIA-232, autrefois connu sous le nom de RS-232, a été développé dans les années 60 par l'association d'industries électroniques (EIA) en tant que standard de communication des données. EIA-232 adresse des liaisons de transmission de données en série et indique le protocole d'échange de données, les tensions et synchronisation de signal, des fonctions de signal, et les connecteurs mécaniques à employer. Les signaux EIA-232 sont asynchrones avec des débits typiques de 20 Kbps. EIA-485 est également un standard des communications séries asynchrones avec des débits typiques de 10 Mbps et la capacité de transmettre des données sur des liens de plus longue distance qu'EIA-232. On l'a autrefois connu comme RS-485 [*AMI00*].

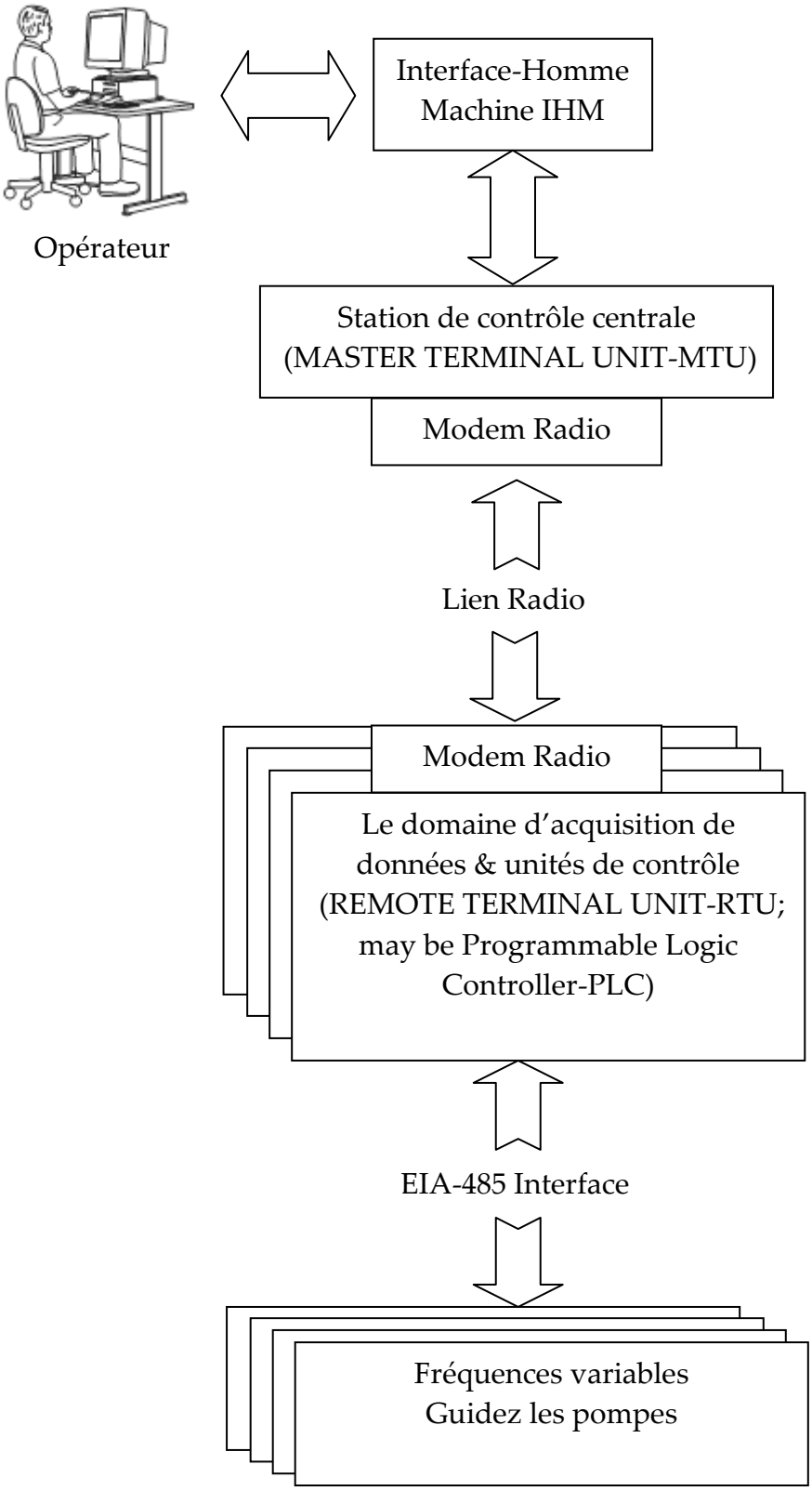


Figure 1.3: Système typique SCADA de pilotage de pompes au gisement de pétrole

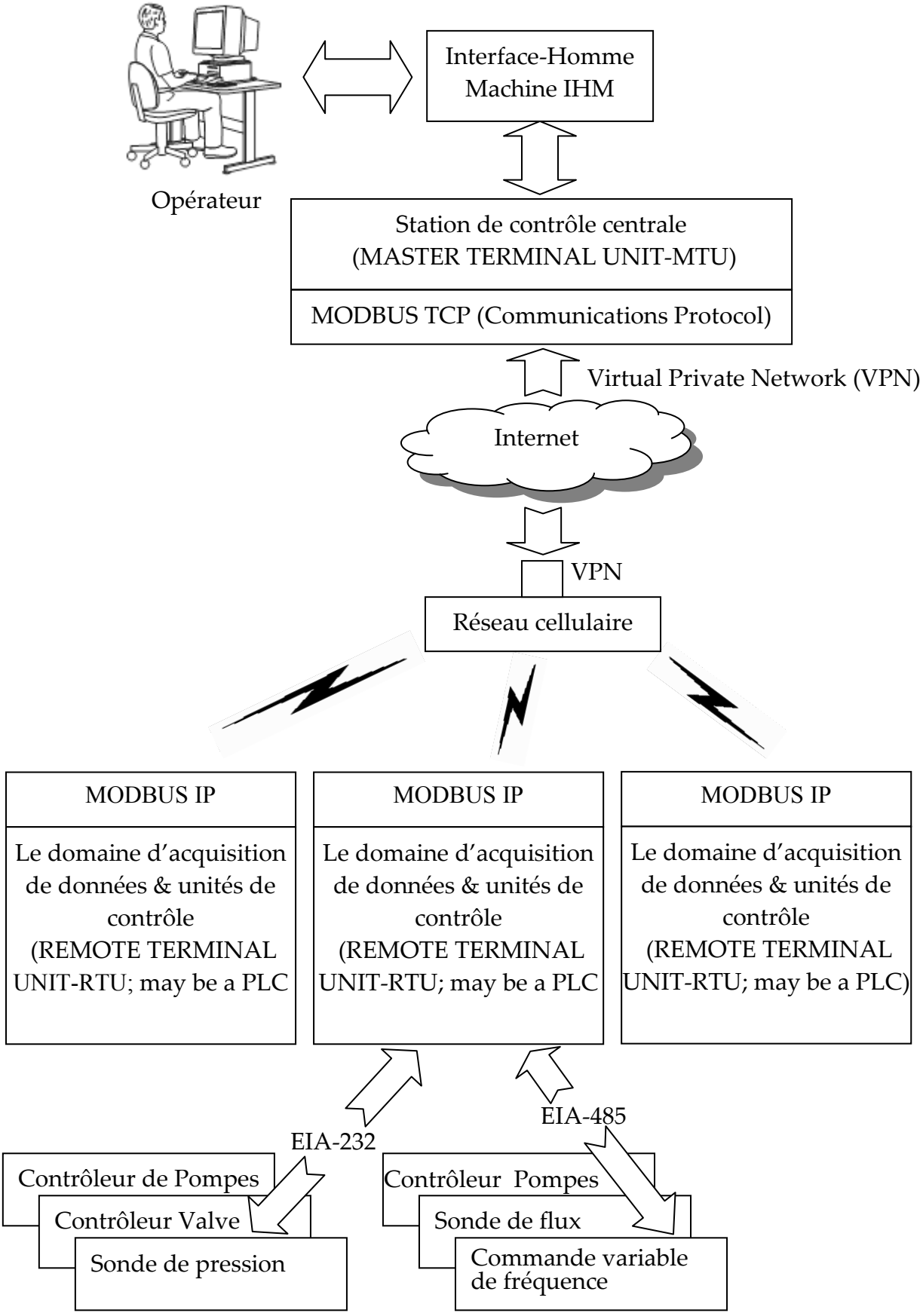


Figure 1.4: Système SCADA utilisant l'Internet et le réseau cellulaire

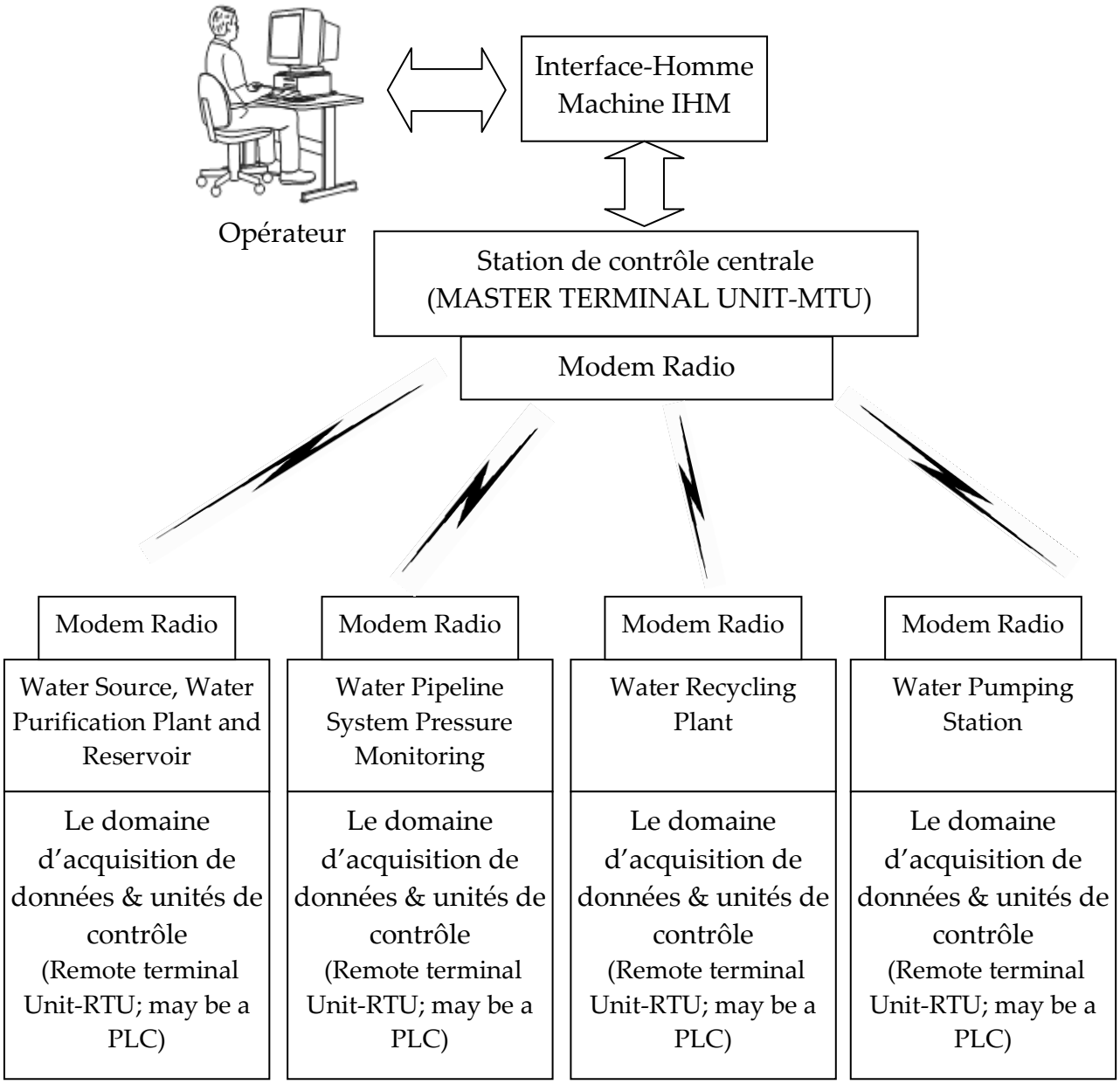


Figure 1.5: Service SCADA de traitement des eaux [*BER02*]

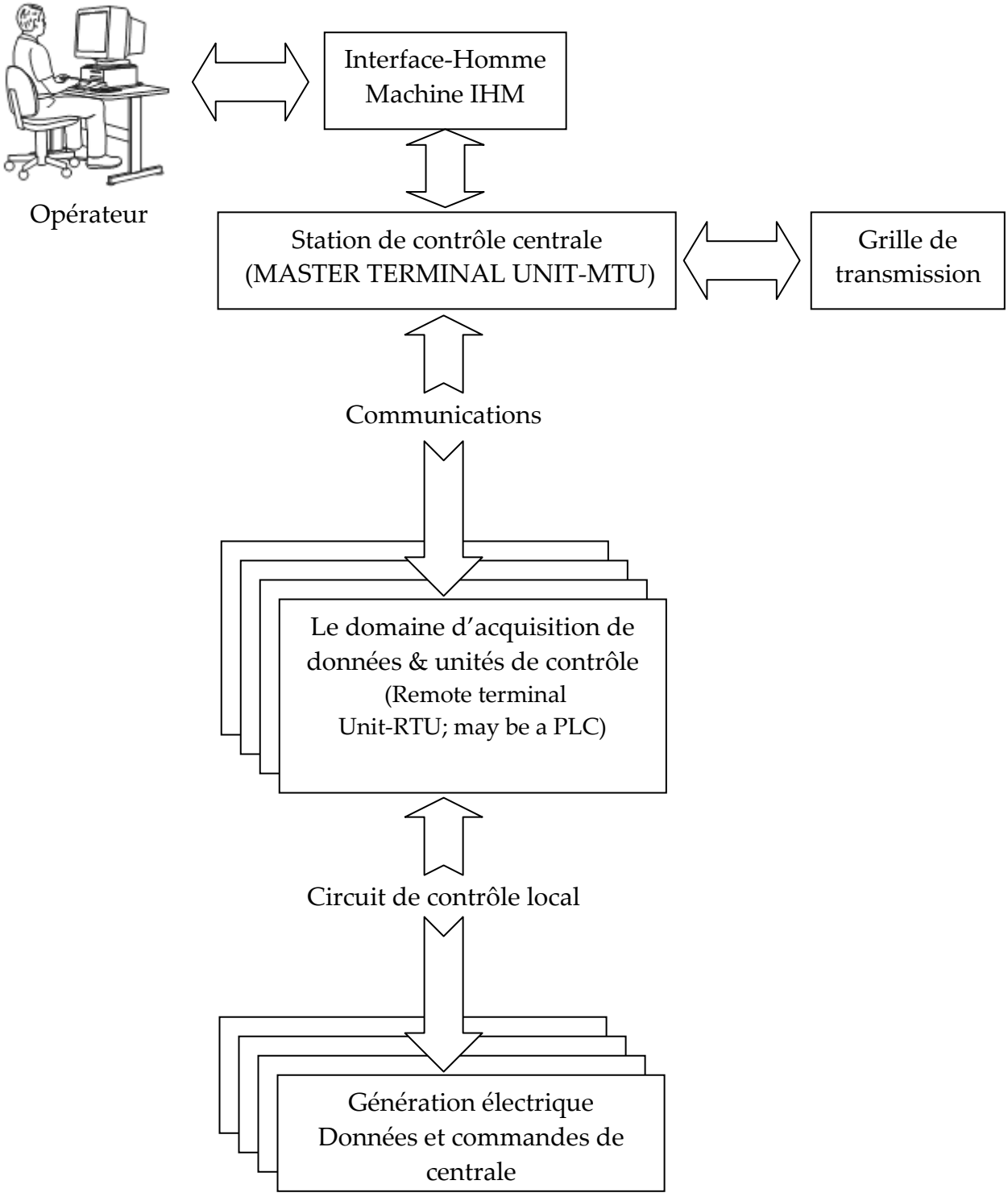


Figure 1.6: Un système SCADA de génération électrique

1.4 Vue d'ensemble sur la sécurité des systèmes SCADA

Pour des raisons économiques, d'efficacité et d'entretien, les plateformes de contrôle et d'acquisition de données ont émigré des réseaux isolés en utilisant le matériel et logiciel de propriété industrielle aux systèmes PC-basés en utilisant les logiciels standards, les protocoles et l'Internet. Le côté incliné de cette transition a été d'exposer les systèmes SCADA aux mêmes vulnérabilités et menaces qui infestent les PCs Windows-basé et leurs réseaux associés. Quelques attaques typiques qui pourraient être montées contre les systèmes SCADA qui utilisent le matériel et le logiciel standard sont classées ici [*CER01*] :

- Code malveillant tel que les virus, Trojan horses et worms
- Divulgation non autorisée des données critiques
- Modification et manipulation non autorisées des données critiques
- Denial of service (DoS)
- Accès et modification non autorisé des journaux d'audit

La plupart des systèmes SCADA, en particulier les PLCs ou les contrôleurs locaux, doivent fonctionner dans les environnements temps-réel ou proches. Ainsi, ils ne peuvent pas avoir de retard qui pourraient être provoqués par les logiciels de sécurité d'informations et qui interfèrent avec les décisions critiques de contrôle affectant la sûreté de personnel, la qualité du produit, et les frais d'exploitation. En outre, les composants industriels d'un système SCADA n'ont pas habituellement une capacité excessive de mémoire qui peut adapter à des programmes relativement grands liés aux activités de surveillance et de sécurité. En résumé, les systèmes conventionnels de la technologie de l'information (IT) sont concernés par la prévision de la connectivité interne et externe, la productivité, les mécanismes étendus de sécurité pour l'authentification et l'autorisation, et les trois principes majeurs de sécurité d'informations à savoir la confidentialité, la disponibilité et l'intégrité.

En revanche, les systèmes SCADA renforcent la fiabilité, la réponse en temps réel, la tolérance en situations de secours où les mots de passe pourraient être inexactement entrés, la sûreté de personnel, la qualité du produit, et la sûreté des installations industrielles.

1.4.1 SCADA et la technologie de l'information (IT)

Il y a une tendance naissante dans beaucoup d'organismes comportant SCADA et des unités de l'IT (Information Technology) vers l'intégration de quelques activités de recouvrement [*BYR02*]. Par exemple, l'ingénierie de contrôle (SCADA property) pourrait être absorbée ou étroitement intégré avec le département d'IT. Cette tendance est motivée par des économies réalisées en consolidant les plateformes, les réseaux, les logiciels, et les divers outils d'entretien. Cette intégration, cependant, vient avec une certaine difficulté. Relative à la sécurité de l'information (ie les architectures de sécurité SCADA et les systèmes d'IT se sont traditionnellement concentrées sur différentes priorités). Un bon exemple de cette sorte de problème est le temps d'arrêt (downtime) habituel programmé par les organismes IT pour améliorer les logiciels, effectuer des sauvegardes, et ainsi de suite. Un tel temps d'arrêt ne peut pas être toléré dans la plupart des systèmes SCADA.

Avec l'intégration des deux systèmes : SCADA et le système d'IT (i.e. emploient le même modèle de sécurité), les issues telles que les modems reliés à un système compromettant l'autre, la possibilité de connexion à l'Internet (exposant le système SCADA aux différentes risques), le temps réel, les besoins déterministes et l'opération round-the-clock des systèmes SCADA exigent le fusionnement des divers connaissances de SCADA et d'IT.

1.4.2 La sécurité Conventionnelle d'IT et SCADA issues

Au cours des années, les professionnels de sécurité des systèmes d'informations ont développé un certain nombre de méthodes acceptées pour protéger les réseaux et les infrastructures de calcul contre les attaques malveillantes. Cependant, ces méthodes ne peuvent pas être appliquées directement aux systèmes SCADA sans prendre en compte les différentes exigences des systèmes SCADA & IT. La liste suivante fournit des exemples des meilleures méthodes d'IT et l'état de leur application aux systèmes SCADA [*BYR02*] :

■■ Audit et contrôle des journaux

L'analyse d'après-le-fait des traces est un moyens utile pour détecter les événements passés. La surveillance, d'une part, implique la capture en temps-réel des données pendant qu'un système fonctionne. Les deux techniques sont utilisées avec succès

dans les systèmes d'IT. Leur application aux systèmes SCADA rapportera des avantages semblables à ceux dérivés de leur utilisation dans les systèmes d'IT. En raison des variations de l'époque et de la sophistication des composants d'un certain système SCADA, beaucoup n'ont pas des possibilités de journalisation. Le coût d'installation, de fonctionnement, de maintenir des capacités de surveillance et de contrôle étendu dans une application SCADA doit être estimé contre les avantages potentiels.

■■ Biométrie

La biométrie est intéressante parce qu'elle utilise un mécanisme d'authentification basé sur une caractéristique physique pour chaque tentative individuelle d'accéder aux composants appropriés d'un système SCADA. Actuellement, la biométrie est prometteuse, mais n'est pas complètement fiable. Selon la caractéristique étant examiné, elle pourrait générer un nombre élevé de rejets erroné ou acceptations incorrects, problèmes de sortie, issues de facteur humain, et compromis possibles du système. Cependant, la technologie progresse et la biométrie devrait devenir une option viable pour contrôler l'accès au système SCADA.

■■ Pare-feu

Les pare-feux peuvent être employés pour masquer le trafic entre une corporation d'IT et un réseau SCADA. Ainsi, dans la plus part de cas, un pare-feu peut protéger les systèmes SCADA contre les pénétrations qui se sont produites du côté de la corporation (i.e. Internet). Quelques issues qui doivent être considérées en appliquant les pare-feux aux systèmes SCADA sont les délias introduits dans les transmissions de données, la compétence et les coûts exigés établis et la gestion des pare-feux, et le manque des pare-feux conçus pour se connecter (se interfacer) à quelques protocoles populaires SCADA.

■■ Systèmes de détection d'intrusion

Les systèmes de détection d'intrusion (IDSs) sont gérés soit par un système central (host-based) ou un réseau (network-based). Un système (IDS) host-based peut détecter des attaques contre le système hôte, mais ne pas surveillent le réseau. Alternativement, un système réseau-basés examine le réseau en surveillant et en évaluant le trafic pour prévenir les actions malveillante. IDSs sont utiles pour protéger les systèmes SCADA, mais ne peuvent pas être universellement appliqués parce que, actuellement, IDSs ne sont pas disponible dans certains protocoles

SCADA. Comme avec d'autres préservations, IDSs pourrait ralentir certaines opérations de SCADA et leur coût et opération doivent être estimés contre les avantages potentiels dérivés de leur utilisation.

■■ Détection et élimination du code malveillant

Le coût de calcule associé à la détection et à l'élimination du code malveillant qui pourrait infecter un système SCADA peut sérieusement affecter l'exécution en temps réel les composants d'un système SCADA. Les activités telles que l'utilisation des antivirus, la mis à jour des bases de données de signature de virus, et la mise en quarantaine ou supprimant le code malveillant exigent des cycles de temps et de calcul qui ne pourraient pas être disponibles sur les composants d'un système SCADA. La mise à jour des bases de données de virus à partir de l'Internet expose également les systèmes SCADA aux virus additionnels et aux attaques du côté Internet. Encore, le coût d'implémentation d'antivirus doit être calculé auprès les risques perçus d'un système SCADA et les avantages d'un tel software.

■■ Mots de passe

Dans un environnement SCADA, un opérateur de contrôle devrait entrer un mot de passe pour accéder à un dispositif en cas d'urgence. Si l'opérateur saisit le mot de passe inexactement plusieurs fois, un paradigme de sécurité, qui présume un intrus essayant de deviner le mot de passe, va interdire l'opérateur d'effectuer son travail. Le rejet de l'opérateur n'est pas une bonne chose dans les environnements de contrôle en temps-réel. Pour des opérateurs sur les dispositifs de contrôle locaux, les mots de passe pourraient être éliminés ou être rendus extrêmement simples. Au niveau de surveillance, de meilleurs et plus longs mots de passe pourraient être employés, l'authentification en deux-facteurs pourrait être utilisée. Dans les situations où les mots de passe pourraient être sujets à l'interception quand sont transmis à travers le réseau, le cryptage devrait être considéré pour protéger le mot de passe contre les risques.

■■ Cryptographie à clé Public

Avec la cryptographie à clé public (asymétrique), il n'y a aucun besoin d'échanger des clés secrètes entre l'expéditeur et le récepteur. Une clé publique est disponible à n'importe qui souhaite communiquer avec le propriétaire de la clé privé correspondante. La clef privée est protégée et connue seulement par le récepteur [*BEL98*]. La caractéristique principale de la cryptographie à clé public est qu'il est

pratiquement impossible de dériver la clé privée de la clé publique connue. La cryptographie à clé Public fournit également la capacité pour qu'un expéditeur signe digitalement un document et le transmettre pour que n'importe qui ait la clé publique peut le lire. Ces signatures garantissent que le document qui a été envoyé par le propriétaire de la clé privée sera lu par sa destination possédant la clé publique. Relativement aux opérations SCADA, les crypto-systèmes à clé public ont besoin des durées du traitement relativement longues qui sont incompatibles avec les conditions en temps réel des systèmes de contrôle.

■■ Cryptographie à clé Symétrique

La cryptographie à clé symétrique, également connu sous le nom la cryptographie de clé secrète, l'expéditeur et le récepteur doit partager une clé commune et secrète. Cette clé est employée pour encrypter le message à l'extrémité de transmission et pour le décrypter à l'extrémité de réception. Ainsi, les clés secrètes doivent être distribuées à l'abri de tous les émetteurs à tous les récepteurs [*BEL98*]. Cette distribution est un souci.

Une solution populaire est d'employer la cryptographie à clé public pour distribuer la clé secrète puis utiliser la cryptographie à clé symétrique pour envoyer un message. Puisque la longueur de la clé est relativement courte comparée au message, le temps n'est pas un problème avec la cryptographie à clé public. La cryptographie à clé Symétrique encore n'a pas été largement appliquée aux systèmes SCADA. Elle est applicable aux données transmises au-dessus d'un réseau SCADA de longue distance et n'est pas importante dans les boucles de contrôle des installations locales. Le cryptage à clé symétrique sera appliqué aux parties critiques d'un réseau SCADA.

■■ Contrôle d'accès basé sur le Rôle

Ce type de contrôle d'accès gagne la popularité dans le gouvernement et les secteurs industriels en raison de sa capacité d'adapter à des changements du personnel et des organismes. Dans ce type de contrôle de sécurité, l'accès est basé sur le rôle d'une personne dans l'organisation plutôt que son identité. Il n'a pas été encore largement appliqué aux systèmes SCADA mais prises des engagements pour l'usage au niveau de surveillance des opérations SCADA.

1.4.3 Redondance comme composante de sécurité en SCADA

En plus des contrôles techniques et administratifs de sécurité, de diverses mesures de sécurité physiques peuvent être appliquées pour protéger les systèmes SCADA. La protection, la réplication, les centres de contrôle géographiquement séparés peut assurer la redondance et, en conséquence, assurer la protection contre les attaques humaines et les catastrophes naturelles. Sur une petite échelle, Un système de réserve (backup) pour SCADA implanté au centre de contrôle supervisé fournit des moyens pour continuer le fonctionnement si le système primaire est handicapé [*BYR00*]. Comme une couche complémentaire de sécurité, le centre de contrôle SCADA a pu être situé dans une région éloignée dans un bâtiment non marqué et inaperçu.

1.5 Propriétés souhaitables d'un système SCADA

La figure **1.7** résume l'état général des systèmes SCADA à ce moment. La figure décrit un système typique SCADA qui incorpore les plateformes matérielles et logicielles standards, telles que les PCs et Windows. Le système SCADA est connecté aux réseaux externes, aux corporations d'IT et aux points d'accès distants, probablement non sécurisés, tels que les modems. Puisque les matériels et les logiciels standards sont employés, les équipements sont vulnérables aux mêmes attaques qui ont été historiquement montées contre des PCs et Windows [*DHS03*].
Une pénétration réussie et non autorisée d'un système SCADA pourrait avoir comme conséquence un intrus prenant le contrôle d'une unité maître ou esclave, perturber des processus critiques, altérant des données, voire lancer des actions qui pourraient avoir comme conséquence la perte de la vie humaine et la destruction de l'industrie sous contrôle [*CIP01*]. Une bonne description des propriétés désirées d'un système SCADA est donnée par NERC (North American Electric Reliability Council) dans la définition des objectifs de fiabilité en SCADA. Quoique la définition adresse l'industrie électrique, les propriétés peuvent être extrapolées à tous les composants de l'infrastructure critique d'une nation. La forme 715M de NERC pour SCADA définit la fiabilité comme :

■■ Adéquation
La capacité de répondre aux spécifications fonctionnelles de système dans des estimations des composantes principales (major component ratings) en présence de

la panne programmée et non-programmée des composants ou des équipements de système

■■ **Sécurité**

Les capacités d'un système pour résister aux perturbations résultants des défauts ou des actions internes ou externes non autorisées sans perte d'équipements et de production et sans dommage de la sûreté humaine.

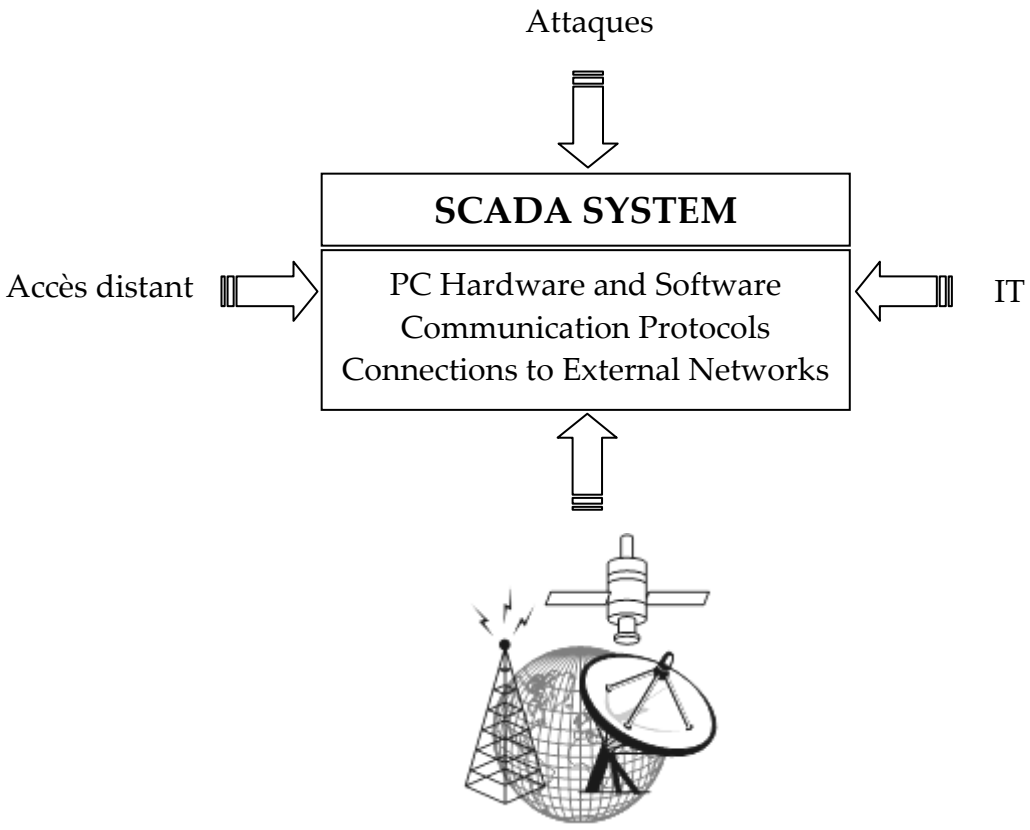


Figure 1.7: Liens typiques d'un système SCADA

1.6 Études de cas

Ici nous listons des incidents récents démontrant la vulnérabilité des systèmes d'infrastructure critiques (SCADA) [*DAC03*]:

■■ Une cascade d'événements en dehors-de-contrôle (black-out) a quitté huit états (usa) et partie du Canada et environ 50 millions de personnes sans électricité en août 2003.

■■ Au printemps 2001, les intrus (hackers) ont pénétré par effraction dans les systèmes informatiques CAL-ISO, opérateur de la grille électrique primaire en Californie, et apparemment n'ont pas été découverts pendant 17 jours.

■■ Grace à un lien sans fil vers un système SCADA en Queensland, Australie, le système de contrôle d'eaux d'égout de Maroochy Shire en 2000 a été exploité par Vitek Boden (ex-employé). Cette attaque a causé des millions de gallons d'eaux d'égout d'être vidés dans des voies d'eau de Maroochy sur une période de quatre mois.

■■ En janvier 2003, un ver de Slammer a dévié le pare-feu du réseau corporatif, neutralisant un système de surveillance de sûreté pour presque cinq heures et l'ordinateur de processus du centre « Plant Process Computer » pendant presque six heures à la centrale nucléaire de l'Ohio Davis-Besse actionnée par First Energy Corp (usa). Un contractant de Davis-Besse qui avait entré dans un réseau non sécurisé avait détourné le ver ''worm'' dans le réseau corporatif interne par l'intermédiaire d'un de plusieurs connections secrètes (obscurément documentés) pour livrer le Slammer-worm. Le point à noter est que le Slammer-worm exploite une vulnérabilité dans le MS SQL Server 2000. Ceci implique que les vulnérabilités dans la plateforme ou l'environnement de fonctionnement sont en fait, hérité par le système SCADA.

1.7 Conclusion

Comme les systèmes SCADA progressaient en architectures autonomes en utilisant le hardware, le software et les protocoles du domaine industriel pour interconnecter ses équipements (comprenant des PCs, OSWindows et des protocoles standards) ils sont également devenus plus vulnérables aux attaques. En outre, les systèmes SCADA sont intégrés dans les systèmes corporatifs de technologie d'information (IT), qui ont différentes caractéristiques de sécurité et de fiabilité que les exigences plus rigoureuses en SCADA.

Nous avons introduit dans ce chapitre les systèmes SCADA en montrons leurs évolution, définitions et architectures de base. Nous avons également présenté quelques méthodes de sécurité pour protéger ces infrastructures critiques. Notre solution de sécurité décrite dans ce rapport est un modèle de pare-feu (en utilisant une méthode dite intrusion-tolerant) et de contrôle de flux d'information, pour cela, ces deux techniques feront l'objet des prochains chapitres pour en parler de manière générale et particulièrement pour les systèmes SCADA.

CHAPITRE

2

La Tolérance d'Intrusion & la Protection des Systèmes SCADA

2.1 Introduction

Il y a un corps important de recherche sur les architectures réparties, ses méthodologies et algorithmes, dans les deux domaines, la tolérance aux fautes et la sécurité. Tandis qu'elles ont pris des voies séparées jusque récemment, les problèmes à résoudre sont de nature similaire. Dans la fiabilité classique, la tolérance aux défauts a été meilleure en beaucoup de solutions. D'un autre côté les travaux sur la sécurité classique ont favorisé, avec moins d'exceptions, la prévention d'intrusion. La tolérance d'intrusion[1] est une nouvelle approche qui a lentement émergé pendant la dernière décennie, et gagne un essor lumineux récemment. Au lieu d'essayer d'empêcher chaque intrusion simple, ceux-ci sont permis, mais tolérés : le système déclenche des mécanismes qui empêchent l'intrusion de produire un échec de sécurité de système.

2.2 Le cas de tolérance d'intrusion

La fiabilité a été définie en tant que propriété d'un système informatique tel que la confiance peut d'une manière justifiable être placée sur le service qu'elle fournit. Le service fourni par un système est son comportement comme il est perceptible par son

[1] Comme exemple approprié de recherche : **MAFTIA** (**M**alicious- and **A**ccidental-**F**ault **T**olerance for **I**nternet **A**pplications), qui a développé des concepts et des architectures; **OASIS** (**O**rganically **A**ssured and **S**urvivable **I**nformation **S**ystem), qui a mis en application plusieurs systèmes intrusion-tolérants.

utilisateur (s) ; un utilisateur est un autre système qui (humain ou physique) agit l'un sur l'autre avec le premier [*AVI01*]. La fiabilité est un corps de recherche qui acquiert un ensemble de paradigmes, parmi lesquels la tolérance aux fautes, et elle s'est développée sous le cadre mental des défauts accidentels, avec peu d'exceptions [*FRA85*, *DOB86*], mais les concepts essentiels peuvent être appliqués aux défauts malveillants d'une façon cohérente

2.2.1 Une brève vue à la tolérance aux fautes et à la sécurité

Les défauts malveillants rendent le problème de la fiabilité d'un système réparti plus dur : des défaillances peuvent n'être plus considérées indépendantes, comme avec les défauts accidentels, tandis que des attaquants sont susceptibles de produire des symptômes « mode-familier » : des composants peuvent réaliser la collusion par des protocoles distribués ; les défaillances eux-mêmes deviennent plus graves, depuis l'occurrence des débits inconsistants , aux mauvais moments, avec une identité (ou un contenu) imitée, peut n'être plus considéré de la « petite probabilité » ; de plus, ils peuvent se produire aux instants ou aux endroits particulièrement incommodes du système, conduits par l'esprit d'un adversaire intelligent. Ceci affecte des dogmes long-crus dans la tolérance aux fautes classique : où les types de défauts et leur distribution suivent statistiquement des modèles définissables ; les propriétés d'environnement peuvent être définies en termes probablement significatifs. Essentiellement, cela vous pouvez compter dessus : les modèles de fautes statiques et les hypothèses d'environnement. La première question qui vient à l'esprit quand adressant la tolérance aux fautes (TF) sous une perspective malveillante est ainsi : *Comment modelez-vous l'esprit d'un attaquant* ?

Maintenant regardons le problème sous une perspective classique de sécurité. Les propriétés typiques de sécurité sont : la confidentialité, ou la mesure dans laquelle un service ou une information est protégé contre la divulgation non autorisée; l'authenticité, ou la mesure dans laquelle un service ou une information est véritable, et protégé ainsi contre la personnification ou le imitation ; l'intégrité, ou la mesure dans laquelle un service ou une information est protégé contre la modification illégitime et/ou non détectée ; la disponibilité, ou la mesure dans laquelle un service ou une information est protégé contre le déni de la disposition ou de l'accès autorisée. Le but d'une conception saine de système est de sécuriser toutes ou certaines de ces propriétés. Traditionnellement, la sécurité a évolué comme combinaison de : empêcher certaines attaques de se produire; enlever les

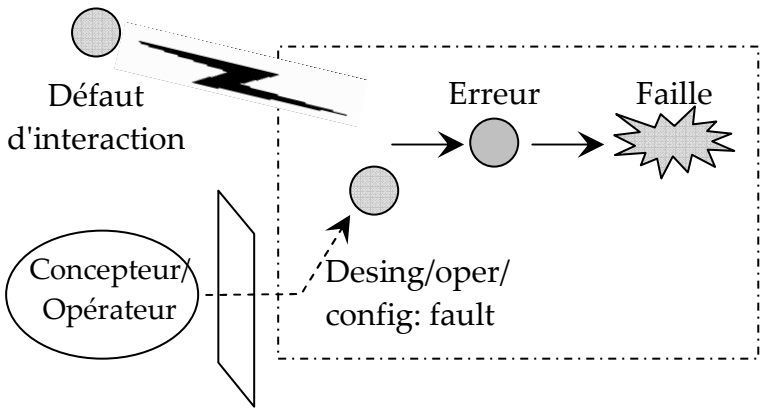
vulnérabilités du logiciel au commencement fragile; empêcher les attaques de mener aux intrusions. Par exemple, afin de préserver la confidentialité, il serait impensable de laisser un intrus lire n'importe quelles données confidentielles du tout. De même, l'intégrité supposerait que ne laissant pas un intrus modifier des données du tout. C'est-à-dire, avec peu d'exceptions, la sécurité a été longtemps basée sur le paradigme d'empêchement.

Cependant, on peut imaginer à titre d'essai le paradigme de tolérance dans la sécurité [*ADE02*] :

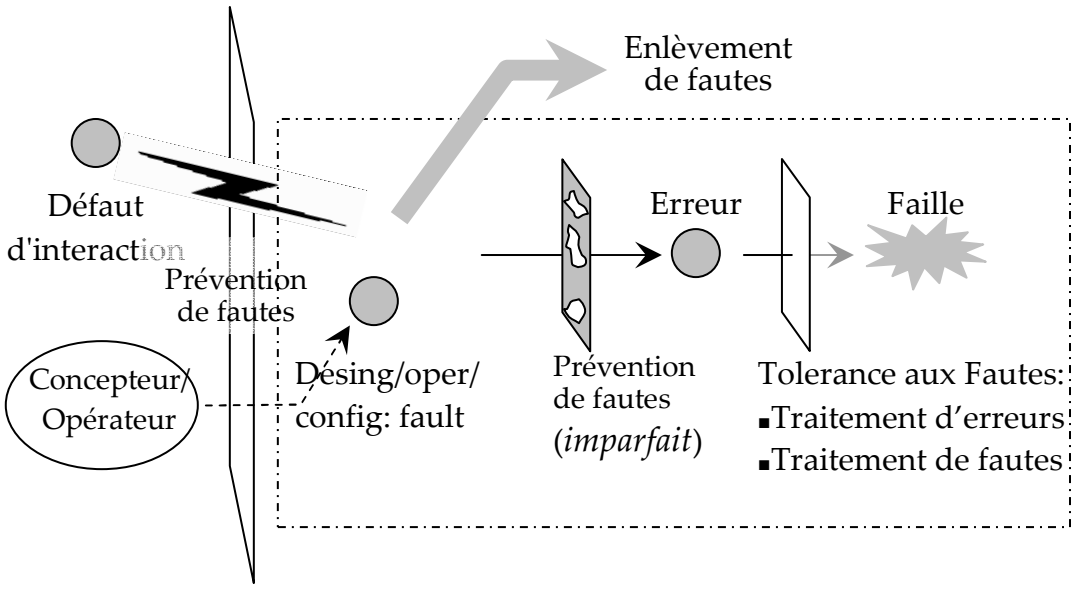
- Supposer (et accepter) que les systèmes demeurent dans une certaine mesure vulnérables ;
- Assumer (et accepter) que les attaques sur des composants/sous-systèmes peuvent se produire et seront réussis ;
- S'assurant que le système global demeure sécurisé et opérationnel. (Puis, une autre question peut être posée : Comment nous laissons des données être lues ou modifiées par un intrus, et assurons toujours la confidentialité ou l'intégrité ?)

2.2.2 Fiabilité comme cadre commun

Observons la séquence bien connue de défaut-erreur-faille sur la figure **2.1**. La fiabilité vise à empêcher l'échec du système. Cet échec a une cause à distance, qui est un défaut (par exemple un bug dans un programme, une erreur de configuration) qui, si activé (par exemple l'exécution du programme passe par la ligne erronée du code), mène à une erreur dans l'état de système. Si rien n'est fait, la faille se manifestera dans le comportement de système.



(a)



(b)

Figure 2.1 (a&b): La séquence : Défaut → Erreur → Faille

En conséquence, la réalisation de la fiabilité implique l'utilisation des combinaisons de : la prévention de défauts, ou comment empêcher l'occurrence ou l'introduction des fautes ; l'enlèvement des défauts, ou comment réduire la présence (nombre, gravité) des fautes ; la prévision de défauts, ou comment estimer la présence, la création et les conséquences des fautes ; et enfin et surtout, la tolérance aux fautes, ou comment assurer la continuité et la disposition correcte de service en dépit des défauts. Ainsi, la réalisation de la fiabilité vis-à-vis des défauts malveillants (par exemple des attaques et des vulnérabilités) signifiera l'utilisation combinée des techniques classiques de prévention et d'enlèvement avec les techniques de tolérance. Analysons les bases de la tolérance aux fautes modulaire et distribuée, et nous voyons comment elles s'adaptent dans ce nouveau scénario :

- Séparation topologique, visant à réaliser l'indépendance des défauts et la dégradation bénévole.
- Réplication, comme forme de redondance, logiciel ou matériel, avec une granularité plus gros ou plus fin.
- Configuration, à travers le nombre de défauts assumés, le nombre de répliques, le type de contrôle d'une réplique (actif, semi-actif, passif, round robin, discipline de vote).

- Optimisation de ressources, par la liaison (mapping) des composants logiciels-aux-matériels, et par l'accroissement des classes F/T (omissive, péremptoire, arbitraire).

La séparation topologique rend la vie de l'intrus plus difficile, en termes d'effort d'attaque. Par exemple, un secret peut être dédoublé par plusieurs emplacements, et obtenir une partie de lui n'indique rien au sujet du tout. La réplication rend les composants plus résilients à endommager, en termes d'intégrité et disponibilité, mais également bénéficient la confidentialité et l'authenticité, si nous pensons à l'exécution répliquée des décisions (i.e. Si pour remettre une fraction de données, ou pour effectuer une authentification. Par exemple, un seul emplacement contaminé (i.e. moniteur de référence) ne pourra pas décider l'autorisation d'accès à une fraction de données.
La configuration est principale pour réaliser la résilience progressive aux différents forces et genres d'intrusions (f+1 répliques pour résister aux attaques de déni-de-service (DoS), 3f+1 pour la résilience aux défauts byzantins, et ainsi de suite), et pour réaliser différents types de rétablissement (actif : le plus prompt, mais également semi-actif et passif). L'optimisation de ressource réduit le coût pour réaliser la tolérance d'intrusion.

2.3 Concepts de la tolérance d'intrusion

Quelle est la tolérance d'intrusion ? Comme dit plus tôt, le paradigme de tolérance dans la sécurité : suppose que les systèmes demeurent dans une certaine mesure vulnérables ; suppose que les attaques sur des composants ou des sous-systèmes peuvent se produire et certains seront réussis ; s'assure que le système global néanmoins demeure sécurisé et opérationnel, avec une probabilité quantifiable. En d'autres termes :

- Les défauts —malveillants et tout autre — se produisent ;
- Ils produisent des erreurs, c.-à-d. les composants de niveau sécurité peuvent être compromis;
- Les mécanismes de traitement des erreurs s'assurent que l'échec de sécurité est empêché.

Évidemment, une approche complète sera combinée la tolérance avec la prévention, l'enlèvement et la prévision.

2.3.1 Un modèle de défauts composite (AVI)

Les mécanismes de la faille d'un système ou d'un composant, sécurité-sensée, doivent réaliser avec une richesse des causes, qui s'étendent des défauts internes (par exemple vulnérabilités), aux externes, les défauts d'interaction (par exemple, attaques), dont la combinaison produit des défauts qui peuvent directement mener à l'échec des composants (par exemple, intrusion). Une intrusion a deux causes fondamentales :

Vulnérabilité : une faute dans un système de calcul ou de communication qui peut être exploité avec l'intention malveillante

Attaque : une faute intentionnelle malveillante injecté dans un système de calcul ou de communication, avec l'intention d'exploiter une vulnérabilité dans ce système qui mène à une :

Intrusion : une faille opérationnelle malveillante résultant d'une attaque réussie sur une vulnérabilité

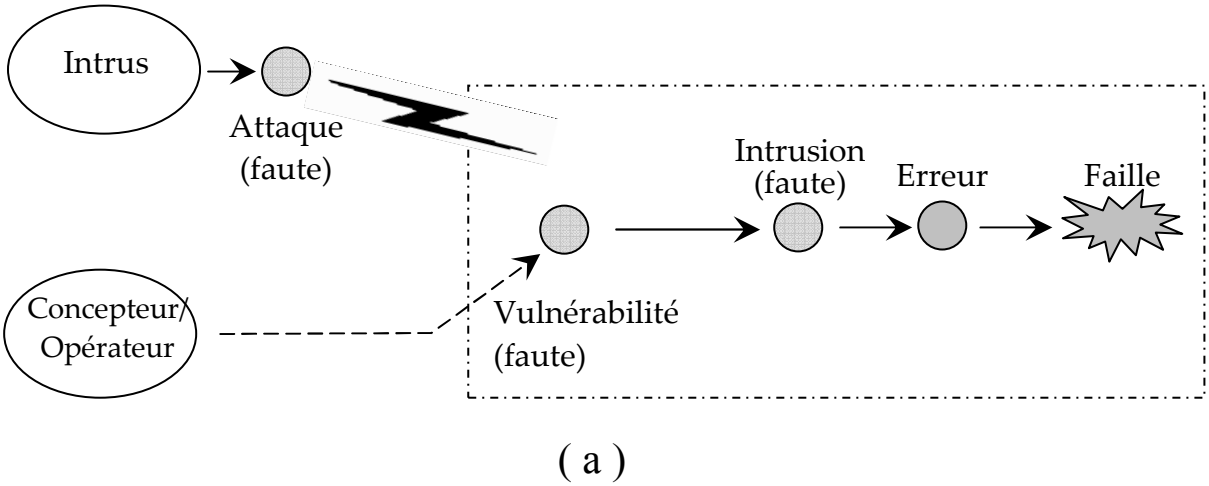
Il est important de distinguer les multiples genres de défauts susceptibles de contribuer à un échec de sécurité. La figure **2.2a** représente l'ordre fondamental de ces trois genres de défauts : **attaque** → **vulnérabilité** → **intrusion** → **faille**. Est ce défini bien la relation entre l'attaque/vulnérabilité/intrusion ce que nous appelons l'AVI modèle de défaut composé. L'ordre d'AVI peut se produire périodiquement dans une série d'événements cohérente produits par l'intrus, également appelé une campagne d'intrusion.

Par exemple, une vulnérabilité donnée a pu avoir été présentée au cours d'une intrusion résultant d'une attaque réussie précédente.

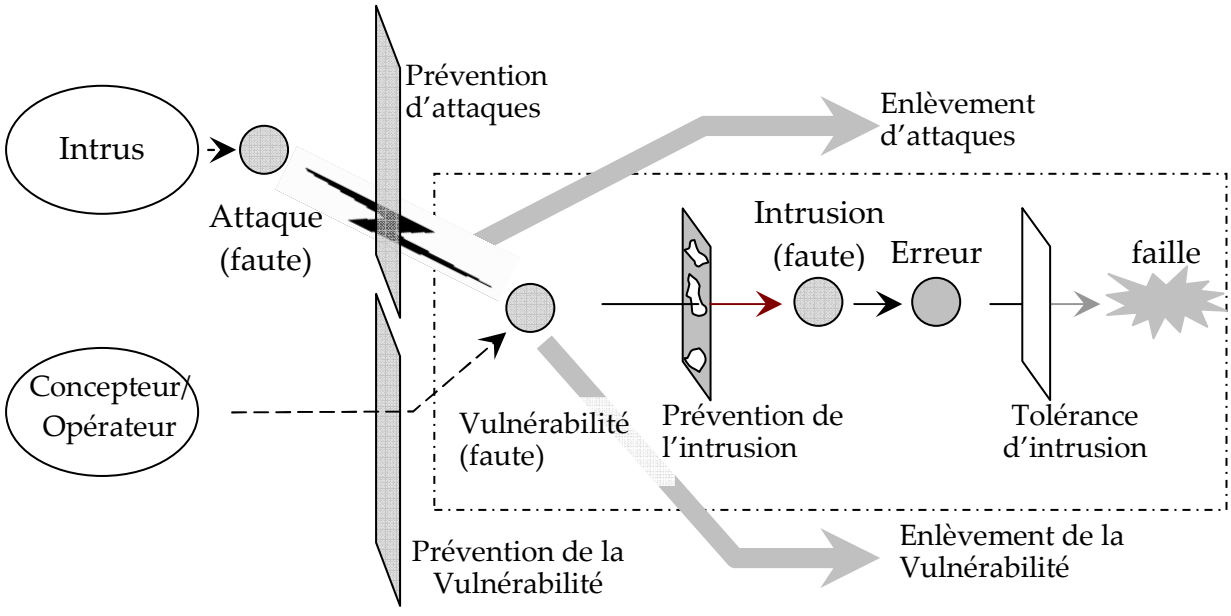
Les vulnérabilités sont les défauts primordiaux existant à l'intérieur des composants, essentiellement les défauts d'exigences, de spécifications, de conception ou de configuration (par exemple, le codage des défauts permettre le débordement de la pile du programme, mots de passe naïfs, ports TCP/IP non protégés). Ce sont habituellement accidentels, mais peuvent être dus des actions intentionnelles, comme précisé dans le dernier paragraphe. Les attaques sont des défauts d'interaction qui essayent avec malveillance d'activer un ou plusieurs de ces vulnérabilités (par exemple, scanne des ports, virus d'email, Java applets Malveillants ou ActiveX controls).

L'événement d'une attaque réussie activant une vulnérabilité s'appelle une intrusion. Cette étape, ultérieurement mène à l'échec du système, est normalement caractérisée par un état incorrect dans le système qui peut prendre plusieurs formes (par exemple, un compte privilégié non autorisé avec un accès telnet, un dossier système avec des autorisations d'accès anormales à l'intrus).

La tolérance d'intrusion signifie que ces erreurs peuvent par exemple être dévoilées par une détection d'intrusion, et elles peuvent être récupérées ou masquées. Cependant, si rien n'a fait pour traiter les erreurs résultant d'une telle intrusion, la faille de certains ou de plusieurs propriétés de sécurité se produira probablement.



(a)



(b)

Figure 2.2: (a) le Modèle de défauts composite (AVI); (b) Prévention de la faille de sécurité

Pourquoi un modèle composé? Le modèle AVI est une spécialisation de la séquence générique défaut → erreur → faille, qui a plusieurs caractères. Premièrement, il décrit le mécanisme d'intrusion avec précision : sans attaques assorties, une vulnérabilité donnée est inoffensive; sans vulnérabilités ciblées, une attaque est non pertinente. Deuxièmement, elle fournit des conseils constructifs pour construire de la fiabilité contre les défauts malveillants, par l'introduction combinée de plusieurs techniques. Pour commencer avec, nous pouvons empêcher quelques attaques d'avoir lieu, en réduisant le niveau de menace, comme indiqué dans la figure **2.2b.**

La prévention d'attaque peut être exécuté, par exemple, en protégeant le dossier de mot de passe dans UNIX, en le rendant indisponible aux lecteurs non autorisés, ou en filtrant l'accès aux parties du système (par exemple, si un composant est derrière un pare-feu et ne peut pas être accédé de l'Internet, l'attaque à partir de là est empêchée). Nous pouvons également effectuer *l'enlèvement d'attaque*, qui consiste à prendre des mesures pour interrompre des attaques en cours. Cependant, il est impossible d'empêcher toutes les attaques, ainsi la réduction du niveau de menace devrait être combinée avec la réduction du degré de vulnérabilité, à travers la prévention de vulnérabilité, par exemple en employant des meilleurs méthodes dans la conception et la configuration des systèmes, ou par l'enlèvement de vulnérabilité (c.-à-d., débogage, patching, neutralisation de modules, etc.) par exemple il n'est pas possible d'empêcher l'attaque (s) qui activent une vulnérabilité donnée. La totalité des techniques mentionnées ci-dessus préfigure ce que nous appelons la prévention d'intrusion, c.-à-d. la tentative d'éviter l'occurrence des défauts d'intrusion.

La figure **2.2b** suggère, comme nous avons discuté plus tôt, qu'il est impossible ou infaisable de garantir une prévention parfaite. Les raisons sont évidentes : il peut être non possible de manipuler toutes les attaques, probablement parce que ne sont pas toutes connues ou les neuves peuvent apparaître ; il peut ne pas être possible d'enlever ou empêcher l'introduction de nouvelles vulnérabilités.
Pour ces intrusions qui encore s'échappant au processus de prévention, des formes de tolérance d'intrusion sont exigées, suivant les indications de la figure, afin d'empêcher l'échec de système. Comme sera expliqué plus tard, ceux-ci peuvent assumer plusieurs formes : détection (par exemple, l'activité de dévoilement de comptes, l'activité de Trojan-Horses); rétablissement (par exemple, interception et neutralisation d'activité de l 'intrus); ou masquer (par exemple, mécanisme de vote entre plusieurs composants, y compris une minorité imposée).

2.3.2 Assurance et séparation des soucis

Analysons comment établir une confiance[2] justifiée sous le modèle AVI. Supposez que le composant C a le prédicat P qui se tient avec une assurance Pr, et ceci définit la fidélité[3] du composant, ⟨*P, Pr*⟩. Un autre composant B devrait ainsi faire confiance à C (jusqu'au degré de C possédant P avec une probabilité Pr). Ainsi, il peut y avoir l'échec conformé à la fidélité limitée de C (c.-à-d., que Pr < 1) : ceux-ci sont « normales », et qui (n'importe quel) dépende de C, comme B, devrait se rendre compte de ce fait, et lui prévoit (et prenne peut-être les provisions pour tolérer le fait dans une perspective plus large de système).

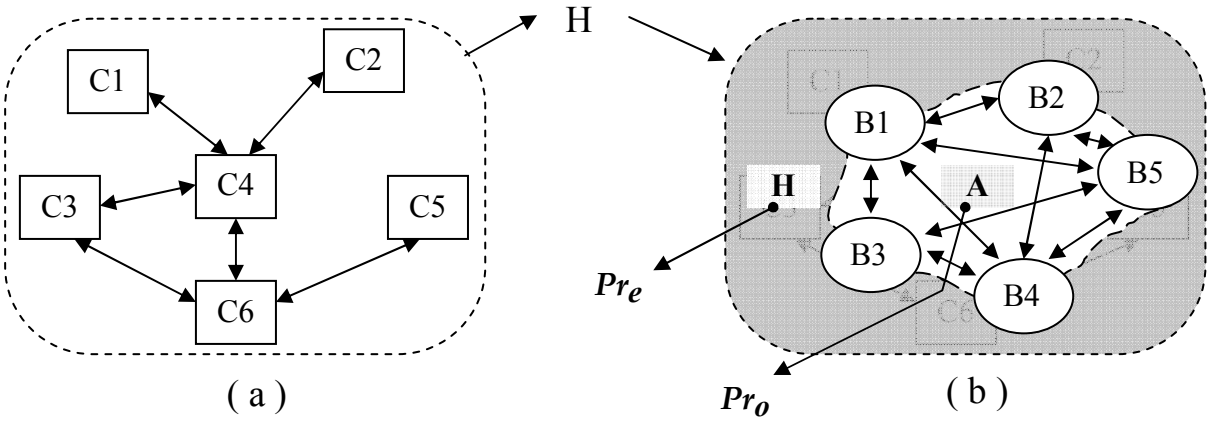


Figure 2.3: Construction de confiance (trust)

Cependant, il peut se produire que B fait confiance à C jusqu'à un plus grand degré qu'il devrait : la confiance a été placée sur C jusqu'à un degré plus grand que sa fidélité, peut-être en raison d'une malle (ou négligée) perception du dernier. C'est une erreur de n'importe quel qui emploie ce composant, qui peut mener aux échecs inattendus.

Enfin, il peut se produire que l'hypothèse faite concernant la fidélité de C est erronée (concernant le prédicat P, ou son assurance Pr, ou tous les deux). Dans ce cas, même si B fait confiance à C jusqu'au degré du ⟨*P, Pr*⟩ il peut également être des failles inattendues (due à une erreur de celui qui a conçu le composant).

---

[2] Confiance : la dépendance admise d'un composant, à l'égard un ensemble de propriétés (fonctionnelles et/ou non fonctionnelles) d'un autre composant, sous-système ou système.

[3] Fidélité : la mesure dans laquelle un composant, un sous-système ou un système, accueille un ensemble de propriétés (fonctionnel et/ou non fonctionnel).

Finalement, que signifie-t-il pour le composant B de faire confiance au composant C ? Il signifie que B assume quelque chose au sujet du C. généralisant, assumant un ensemble B de participants (B_1 à B_n), qui exécutent un algorithme offrant un ensemble de propriétés A, sur un environnement d'exécution (run-time support environment) s'est composé d'un ensemble C de composants (C_1 à C_n). Cette vision modulaire est très adéquate pour les systèmes répartis. Imaginant l'environnement comme représenté dans la figure **2.3a :** C est conçu afin d'offrir un ensemble de propriétés, s'appeler H. Ceci servir d'environnement de soutien sur lequel B fonctionne, comme suggéré par le coussin ombragé dans la figure **2.3b.**
Observant que C confie B pour fournir H : B dépend des propriétés d'environnement H pour mettre en application l'algorithme sécurisant les propriétés A. de même, un utilisateur de B confie le dernier pour fournir A. Sans plus de discours, cette chaîne de confiance serait : si C est confié pour fournir H, alors B est confié pour fournir A.
Maintenant on observe le côté de fidélité. H se tient avec une probabilité Pr_e, l'assurance environnementale de l'hypothèse [*POW92*] :
$Pr_e = \Pr(H \mid f)$, où f représente toute faute

Pr_e mesure la fidélité de C (pour sécuriser les propriétés H). Etant donné H, A a une certaine probabilité (peut être 1 si l'algorithme est déterministe et correct, peut être moins de 1 s'il est probabiliste, et/ou s'il a des défauts de conception) d'être accompli, l'assurance Pr_o (ou l'assurance opérationnelle de l'hypothèse) : $Pr_o = \Pr(A \mid H)$
Pro mesures la confiance sur B sécurisant les propriétés A (étant donné H comme environnement). Puis, la fidélité du composant individuel B (pour sécuriser les propriétés A étant donné H) serait donnée par Pr_o.
ces équations devraient placer des limites sur l'ampleur des relations de confiance. B devrait faire confiance à C jusqu'au degré de fournir H avec confiance $\Pr e \leq 1$.
Cependant, puisque la confiance de l'utilisateur sur B est impliquée par la confiance du B sur C, alors l'utilisateur ne devrait pas confier B en isolation, mais conditionné à la fidélité du C, c'est-à-dire, jusqu'au degré de fournir A avec confiance :
$\Pr_a = \Pr_o \times \Pr_e = \Pr(A \mid H) \times \Pr(H \mid f) = \Pr(A \mid f)$, où f : toute faute

La chaîne résultante a pu continuer périodiquement (récursivement). $\Pr_a$ est la probabilité qu'un utilisateur du système composé de B et de C apprécie les propriétés A, en d'autres termes, il mesure sa fidélité.

2.4 Les bordures et mécanismes de tolérance d'intrusion

Après présentation des concepts de tolérance d'intrusion, nous commençons cette section en analysant brièvement les cadres principaux avec lesquels l'architecte peut travailler afin d'établir des systèmes tolérants d'intrusion : communication sécurisée et insensible aux défaillances ; tolérance d'intrusion basée sur le logiciel; tolérance d'intrusion basée sur le matériel. Nous regarderons également plusieurs cadres connus de sécurité [*VER01*] sous une perspective de TI (Tolérance d'Intrusion). Alors nous revoyons quelques mécanismes de traitement d'erreurs afin d'enlever des intrusions.

2.4.1 Une communication sécurisée et tolérante aux intrusions

C'est le cadre concernant le corps des protocoles assurant une communication tolérante aux intrusions. Essentiellement, sont des canaux sécurisés et enveloppes sécurisés, et des communications insensibles aux défaillances classiques.
Les canaux sécurisés sont normalement établis pour des communications régulières entre les principaux, ou des communications qui durent assez longtemps pour que le concept de session ou de connexion aient un sens. Par exemple, transfert de fichiers ou sessions distantes. Ils vivent sur une différence de résilience/vitesse, parce qu'ils sont en ligne, et peuvent employer des combinaisons d'encryptage physique et virtuel. Les canaux sécurisés adoptent la sécurité par-session, et emploient normalement la communication à cryptage symétrique, et la signature ou l'authentification de canal basée sur la somme cryptographique (MAC).
Les enveloppes sécurisées sont employées principalement pour les transmissions sporadiques, telles que l'email. Ils recourent à la sécurité dite ''par-message'' et peuvent se servir d'une combinaison de la cryptographie symétrique et asymétrique (également appelée hybride) comme une forme d'amélioration des performances, particulièrement pour les messages de grands corps.
Plusieurs techniques aident à la conception des protocoles de transmission insensibles aux défaillances (fault-tolerant). Leur choix dépend de la réponse à la question suivante : Quelles sont les classes des failles des composants du réseau de transmission ?
Pour l'architecte, ceci établit un lien fondamental entre la sécurité et la tolérance de fautes. Dans la communication tolérante aux défaillances classique, il est fréquent de voir les modèles de défaut omissif (accident, omissions, etc.). Dans IT les hypothèses

du mode de défaillance devraient être orientées par le modèle de défaut ''AVI'', et d'ailleurs les propriétés des composants spécifiques peuvent restreindre ce qui devrait être l'hypothèse de départ : faille arbitraire (combinaison du comportement omissif et autoritaire). En fait, c'est le modèle Baseline le plus adéquat pour représenter l'intelligence malveillante.

2.4.2 La tolérance d'intrusion Logiciel-basée

La tolérance de fautes ''Logiciel-basée'' a été principalement visée à tolérer des défauts matériels en utilisant des techniques logicielles. Une autre facette importante est la tolérance de fautes logicielles, destinée à tolérer des défauts de conception logicielle par la diversité de conception. En conclusion, on a longtemps connu que la tolérance de fautes logiciel-basée par la réplication peut également être extrêmement efficace à manipuler les défauts logiciels transitoires et irréguliers [*VER01*].
Même une seule réplique avec des composants homogènes peut donner des résultats significatifs. Comment ? Quand les composants ont une haute fidélité que des hypothèses peuvent être faites concernant la dureté de réaliser une attaque réussie en une vulnérabilité sur l'un d'entre eux (par exemple ''briser un composant''). Dans ce cas, nous pourrions appliquer le principe classique de réaliser une fiabilité d'une réplique établit beaucoup plus élevée que la fiabilité de la réplique individuelle. Par exemple, la réplique simple peut être employée pour tolérer des attaques, en la rendant difficile et prolongée pour que l'attaquant lance des attaques simultanées à toutes les répliques avec succès.

L'introduction d'une vulnérabilité dans un premier temps d'une campagne d'intrusion (être encore exploité par des attaques suivantes) peut également être discutée sous la perspective de la puissance d'attaquant. Cependant, elle peut méthodiquement être présentée d'une manière discrétionnaire, à la différence des attaques qui exigent la synchronisation, et ceci rend de telles attaques potentiellement difficiles à évaluer, et ainsi plus dangereuse que la vulnérabilité originale analysée individuellement.

2.4.3 La tolérance d'intrusion Matériel-basée

La tolérance de fautes Logiciel-basée et matériel-basée ne sont pas des cadres de conception incompatibles [*VER01*]. Dans un contexte modulaire et distribué de systèmes, la tolérance de fautes matérielles aujourd'hui devrait plutôt être vue en

tant que des moyens pour construire les composants dites ''panne-contrôlées'', en d'autres termes, les composants qui sont empêchés de produire certaines classes des failles. Ceci contribue à établir des niveaux améliorés de la fidélité, et d'employer la confiance améliorée correspondante pour réaliser des systèmes tolérants aux défaillances plus efficaces.

Les algorithmes distribués qui tolèrent les défauts arbitraires sont chers en ressources et en temps. Pour des raisons d'efficacité, l'utilisation des composants matériels avec des modes de défaillance contrôlés et imposés est souvent recommandée, en tant que des moyens de fournir une infrastructure où des protocoles résilients à des faille plus bénins peuvent être employés, sans qu'impliquer une dégradation dans la résilience du système aux défauts malveillants

2.4.4 Quelques cadres de sécurité

Dans cette section nous allons Observez que quelques mécanismes concernant les cadres connus dans la sécurité (les canaux sécurisés et les enveloppes, l'authentification, la protection, la communication cryptographique) [*VER01*] peuvent être revisités sous la perspective de tolérance d'intrusion (TI) et constituer ainsi des outils conceptuels utiles pour l'architecte des systèmes tolérants aux intrusions.

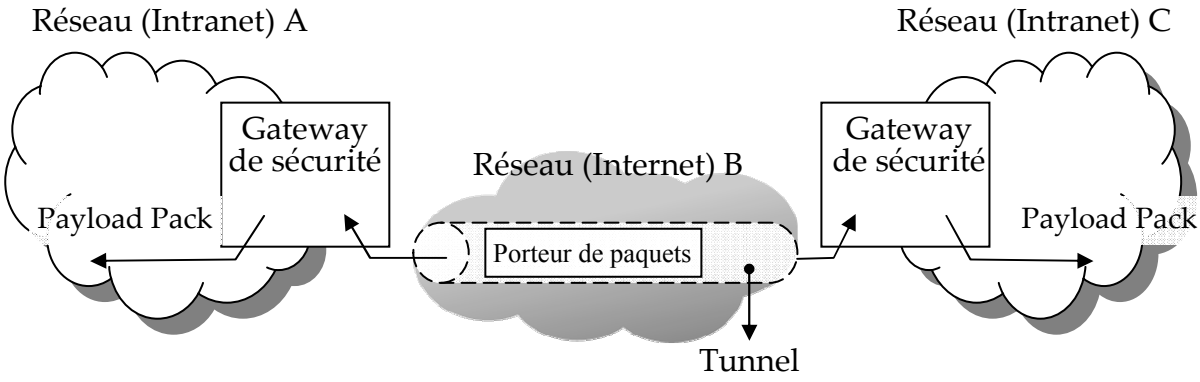


Figure 2.4: Les tunnels, des canaux sécurisés et des enveloppes

Les tunnels sécurisés, par exemple tels construits sur l'Internet avec les canaux sécurisés IP-over-IP, sont des dispositifs de prévention d'intrusion (la figure **2.4**) : ils imposent la confidentialité, intégrité (et parfois authenticité) entre les points d'accès

(gateways), en dépit des tentatives d'intrusion. L'assurance est donnée par : la résilience de la méthode de tunnel, et les passages au point d'accès.

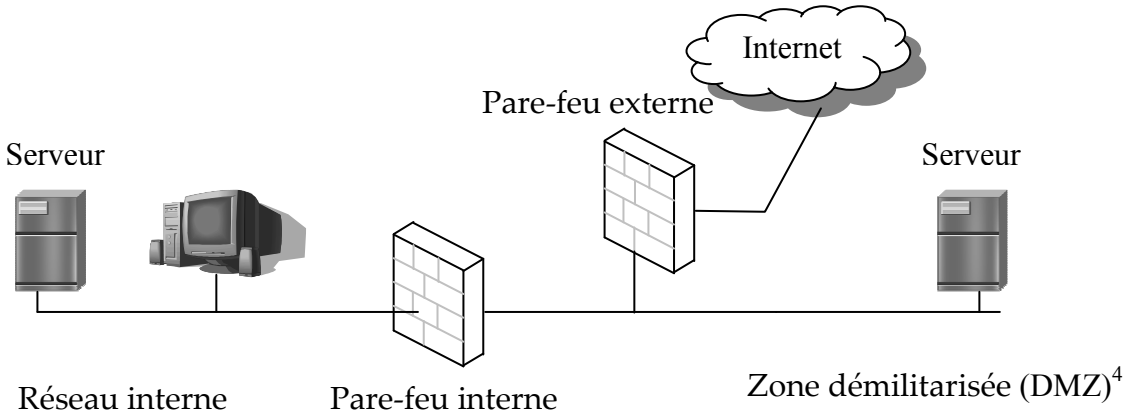


Figure 2.5: Pare-feux (Firewalls)

Les pare-feux sont des dispositifs de prévention d'intrusion (la figure **2.5**) : ils empêchent des attaques auprès les machines dedans qui pourraient exploiter des vulnérabilités menant à une intrusion. Leur assurance est donnée par : la puissance de la sémantique des fonctions de pare-feu, et la résilience des bastions.
Mécanismes et protocoles fournissant l'authentification entre deux entités ou plus (signature, codes d'authentification de message (MAC)) sont également les dispositifs de prévention d'intrusion (figure **2.6a**) : ils imposent l'authenticité empêchant de forger l'identité des participants ou l'auteur/d'origine des données. L'assurance est donnée par : la résilience de la méthode de signature/authentification.

Enfin et surtout, quelques protocoles cryptographiques sont des modules très importants de tolérance d'intrusion qui peuvent être employés périodiquement. Vu comme modules, les protocoles self-enforcing tels que le consensus byzantin ou le multicast atomique (figure **2.6b**), sont des dispositifs de tolérance d'intrusion : ils exécutent le traitement des erreurs ou le masquage (3f +1, 2f +1, f +2, selon le modèle de faute) et assurent la livraison de message en dépit des intrusions réelles.

[4] DMZ est une région qui fournit une séparation entre un réseau externe ou public et un réseau interne ou privé

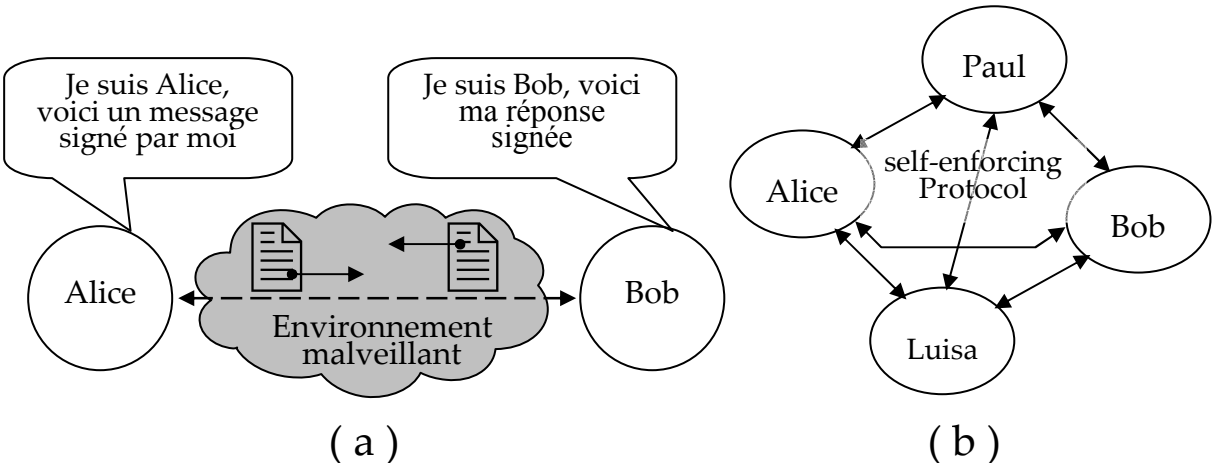


Figure 2.6: (a) Authentification ; (b) Communication et consensus

L'assurance est donnée par : la sémantique des fonctions du protocole, les hypothèses fondamentales du modèle. Les protocoles confiants (TTP : Trusted Third Party) sont également des dispositifs de tolérance d'intrusion qui effectuent le traitement/masquage des erreurs, mais dépendent d'un TTP pour leur opération correcte (figure **2.7a**). L'assurance dépend à : la sémantique des fonctions du protocole, les hypothèses fondamentales du modèle, la résilience de TTP. En conclusion, les protocoles cryptographiques avec seuil (threshold cryptographic protocols) sont des dispositifs de tolérance d'intrusion (figure **2.7b**) : ils effectuent le traitement/masquage des erreurs dans une hypothèse de seuil sans plus que f+1 apparaissent de n intrusions. Leur assurance est donnée par : la sémantique des fonctions cryptographiques, la résilience force-brute du code secret, les hypothèses supposées du modèle.

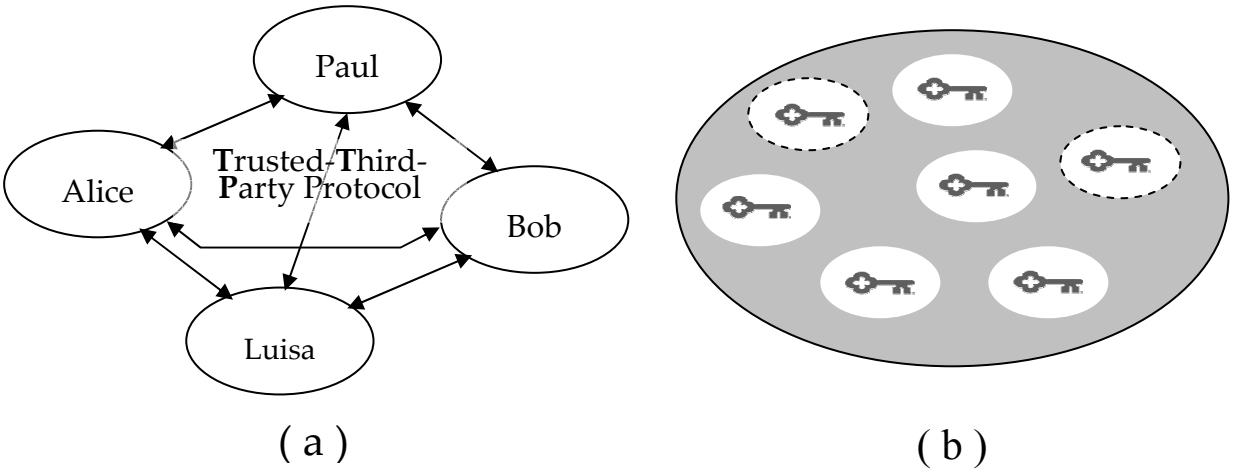


Figure 2.7: (a) Trusted Third Party; (b) cryptographie de seuil

2.4.5 Traitement des erreurs dérivant des intrusions

Après nous passons en revue des classes des mécanismes pour traiter des erreurs dérivant des intrusions. Essentiellement, nous discutons les mécanismes typiques de traitement des erreurs utilisés dans la tolérance de fautes, sous une perspective de tolérance d'intrusion : détection des erreurs ; rétablissement des erreurs et masquage des erreurs.

■■ **La détection des erreurs** est concernée par détecter l'erreur après qu'une intrusion soit activée. Elle vise à : l'emprisonner pour éviter la propagation ; déclenchement des mécanismes de rétablissement d'erreur ; déclenchement des mécanismes de traitement de défaut. Les exemples des erreurs typiques sont : messages forgés ou contradictoires (byzantins); fichiers ou variables mémoires modifiés ; faux comptes d'OS ; sniffers, worms, virus, en opération.

■■ **Le rétablissement d'erreur** est concerné par la récupération de l'erreur une fois qu'il est détecté. Il vise à : fournir des services corrects en dépit de l'erreur ; récupération des effets des intrusions. Les exemples du retour à un état antérieur sont : le système va à un état précédent connu comme correct et se récupère ; le système ayant subi des attaques de déni de service (DoS), ré-exécuter les opérations affectées; le système ayant détecté des fichiers corrompus, fait une pause, les réinstalle, se restaurer au dernier point correct. Le rétablissement vers l'avant peut également être employé : le système procède en avant à un état qui assure la fourniture correcte de service; le système détecte l'intrusion, considère les opérations corrompues comme perdues et augmente de niveau de sécurité (seuil/quorums augmentent, renouvellement de clé); le système détecte l'intrusion, se permute au mode opérationnel dégradé mais plus sûr.

■■ **Masquage d'erreur** est un mécanisme préféré quand, comme se produit souvent, la détection des erreurs n'est pas fiable ou peut avoir une grande latence. La redondance est employée systématiquement afin de fournir le service correct sans problème apparent. Comme exemples : le vote systématique des opérations ; consensus byzantin et uniformité interactive; fragmentation-redondance-dispersion ; corrélation de capteurs (accord sur des valeurs imprécises).

2.5 Une protection intrusion-tolérante pour les infrastructures critiques

Les infrastructures critiques comme l'énergie, l'eau et les réseaux de distribution de gaz ont un rôle fondamental dans la vie moderne. Ces infrastructures sont essentiellement des mécanismes physiques/mécaniques contrôlés électroniquement. Les systèmes de contrôle, habituellement appelés SCADA (contrôle et acquisition de données de surveillance) ou PCS (système de contrôle de processus), se composent des ordinateurs interconnectés par des réseaux informatiques [*MAD05*, *VER06*].

Ces dernières années ces systèmes ont évolué en plusieurs aspects sur lesquels a considérablement augmenté leur exposition aux cyber-attaques venant de l'Internet. Premièrement, les ordinateurs, les réseaux et les protocoles dans ces systèmes de contrôle ne sont plus propriétaires mais des PCs et réseaux standard (par exemple, Ethernet câblé et sans fil), et les protocoles sont souvent encapsulés sur le TCP/IP. Deuxièmement, ces réseaux sont habituellement reliés à l'Internet indirectement par le réseau de corporation ou à d'autres réseaux en utilisant des modems et des liaisons de données. Troisièmement, plusieurs infrastructures sont interconnectées créant une complexité qu'il est difficile de la gérer [*EET06*].

Par conséquent ces infrastructures ont un niveau de vulnérabilité semblable à d'autres systèmes reliés à l'Internet, mais l'impact socio-économique de leur échec peut être énorme. Ce scénario, renforcé par plusieurs incidents récents [*WIL06*, *BYR04*], produit d'un grand souci concernant la sécurité de ces infrastructures, particulièrement au niveau de gouvernement. Une architecture de référence a récemment proposé pour protéger les infrastructures critiques, dans le contexte du projet de CRUTIAL[5] EU-IST [*VER06*]. L'architecture entière d'infrastructure est modelée comme un WAN-de-LANs. La figure **2.8** illustre l'utilisation de CIS (**C**RUTIAL **I**nformation **S**witch) protégeant plusieurs LANs d'une infrastructure critique.

[5] **CRUTIAL**: **CR**itical **UT**ility **I**nfrastructur**AL** resilience: http://crutial.cesiricerca.it.

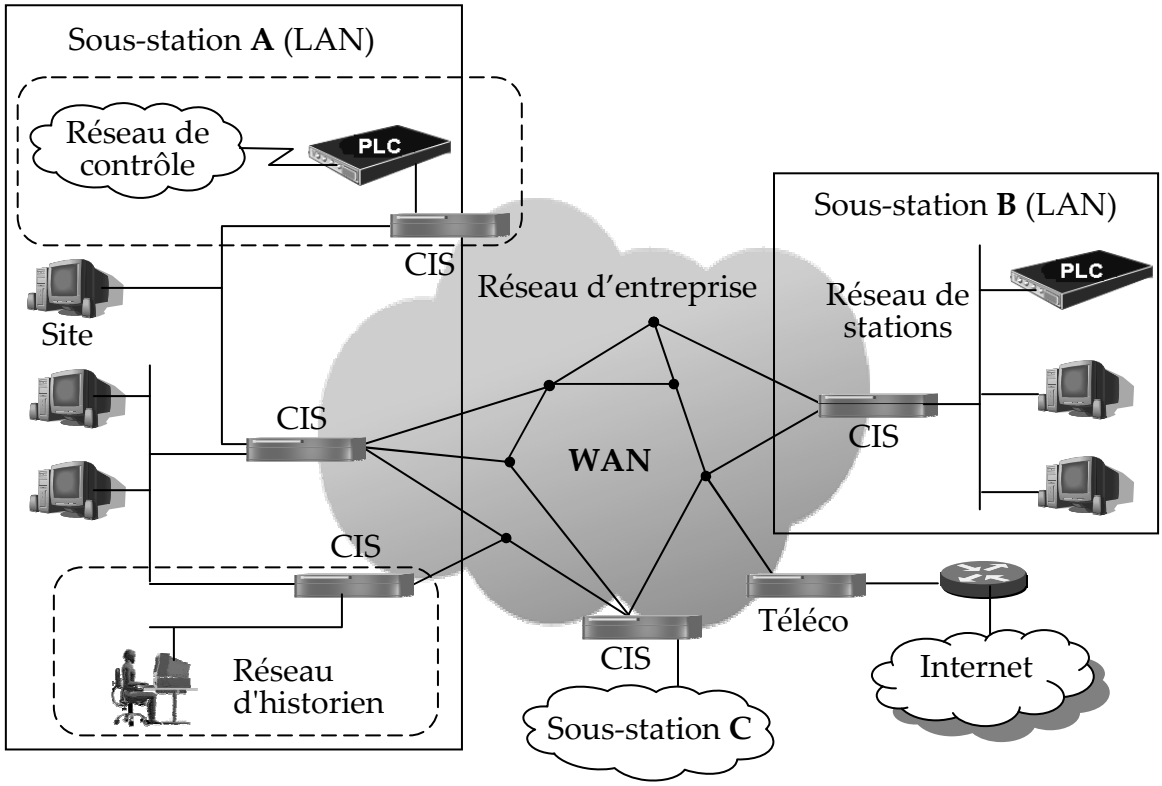


Figure 2.8: WAN-de-LANs connecté par CIS.

Cette topologie permet des solutions simples aux problèmes durs tels que les sous-réseaux de contrôle de legs, et l'interconnexion du trafic critique et non critique. Typiquement, une infrastructure d'information critique est constituée par des équipements, comme les sous-stations de transformation d'énergie ou des agences de corporation, modelée comme collections de LANs et reliée ensemble par un réseau de large-secteur, modelé comme WAN, dans le modèle WAN-de-LANs.

Cette architecture laisse définir des royaumes avec différents niveaux de fidélité. Ce modèle s'intéresse par le problème de protection des royaumes les uns des autres, c.-à-d., un LAN contre un autre LAN ou contre le WAN. Cependant, donné la facilité de définir des LANs dans des architectures IP d'aujourd'hui (i.e. par Virtual switched LANs), il n'y a pratiquement aucune restriction au niveau de la granularité des domaines de protection, qui peuvent descendre à un site simple. En conséquence, ce modèle et cette architecture nous permettent de traiter les menaces de l'extérieur (protégeant un service contre l'Internet) et les menaces de l'intérieur (protégeant un hôte critique contre d'autres hôtes dans la même installation physique, en les localisant dans des LANs différents). La protection de LANs contre le WAN ou

d'autres LANs est faite par un dispositif appelé le commutateur d'information CRUTIAL (CIS)[6].

2.5.1 Le service de protection CIS

Le CIS travaille spécialement comme un pare-feu. Il capture les paquets qui lui traversent, vérifie s'ils satisfont la politique de sécurité étant imposée, expédie les paquets approuvés et rejette ceux qui ne satisfont pas la politique. Cependant, plusieurs autres caractéristiques du CIS lui font un dispositif unique de protection. Ces caractéristiques sont présentées dans cette section.

■■ Pare-feu distribué

CIS peut être employée dans une manière redondante, l'application des mêmes politiques dans différents points du réseau. Un cas extrême dans le côté de SCADA/PCS est d'avoir un CIS dans chaque point d'accès (gateway) interconnectant chaque réseau de sous-station (LAN), et un CIS pour protéger spécifiquement chaque composant critique du réseau SCADA/PCS. Le concept est similaire à employer des pare-feux pour protéger des hôtes au lieu seulement des frontières de réseau [*BEL99*], et qui est particulièrement utile pour les infrastructures d'information critiques donnant leur complexité et criticalité, avec beaucoup de chemins dans le réseau de contrôle qui ne peut pas être facilement bloqués (par exemple, Internet, modems commutés, VPNs, points d'accès sans fil) [*BYR04*].

■■ Pare-feu de niveau application

Les infrastructures critiques ont beaucoup de composants d'enchaînement qui ont été conçus sans sécurité à l'esprit, ainsi n'utilisent pas des mécanismes de sécurité comme le contrôle d'accès et la cryptographie [*DZU05*]. Puisque ces mécanismes de sécurité ne sont pas une partie des protocoles et des systèmes SCADA/PCS, qui doivent encore être protégés, la protection doit être déployée dans un certain point entre l'infrastructure et les hôtes qui lui accèdent. Le CIS doit inspecter et évaluer les messages considérant la sémantique de niveau application parce que, comme déjà dit, l'application (infrastructure) elle-même ne la vérifie pas.

[6] Un CIS fournit deux services de base : le service de protection (PS : s'assure que le trafic entrant et sortant du in/out du LAN satisfait la politique de sécurité de l'infrastructure) et le service de communication (CS : support les communications sécurisées entre CIS et, finalement, entre LANs.).

■■ Un modèle riche de contrôle d'accès

Sans compter que la capacité d'inspecter les messages de niveau application, les CIS ont besoins de soutenir une politique riche de contrôle d'accès qui tient compte de la nature multi-organisationnelle des infrastructures critiques aussi bien que leurs différents états opérationnels. Prenant le système d'alimentation électrique comme exemple, il y a plusieurs compagnies impliquées dans la génération, la transmission et la distribution de l'énergie, comme des organismes de régularisation, et plusieurs de ces parties peuvent exécuter des opérations dans la grille d'énergie. D'ailleurs, presque toutes les opérations de système d'alimentation sont basées sur un modèle d'état classique de la grille [*LHF78*]. Dans chaque état de ce modèle, des mesures spécifiques doivent être prises (par exemple, des actions déterminées dans un plan de défense, pour éviter ou récupérer une interruption d'énergie) et plusieurs de ces actions ne sont pas permises dans d'autres états (par exemple, un générateur ne peut pas être séparé automatiquement quand la grille est dans son état normal). Ces deux facettes complexes de contrôle d'accès en infrastructures critiques exigent des modèles plus élaborés que le contrôle d'accès obligatoire (MAC), discrétionnaire (DAC) ou basé sur le rôle (RBAC).

Pour traiter ça, l'architecture CRUTIAL adopte un modèle plus élaboré, OrBAC[7] (contrôle d'accès basé sur l'organisation) [*AEK03*]. Il permet la spécification des politiques de sécurité contenant des permissions, prohibitions, engagements et recommandations, tenant compte le rôle du sujet, qui fait partie d'une organisation, l'action qu'il veut exécuter, l'objet cible de cette action, et le contexte dans lequel elle est exécutée. Un exemple : dans contexte d'urgence, les opérateurs de la compagnie C peut exécuter des opérations d'entretien sur le dispositif D.

■■ Pare-feu tolérant aux Intrusions

Le niveau de la sécurité des systèmes courants reliés à l'Internet n'est pas proportionné pour les infrastructures que nous sommes concernés avec, donné leur criticalité. Pour améliorer la sécurité et la fiabilité du CIS, il est conçu pour être intrusion-tolérant [*VER03*] : il est replié dans ***n*** machines et suit ses spécifications tant que au plus f de ces machines sont attaqués et font corrompre leur comportement. Pour augmenter la résilience du CIS, une stratégie de rajeunissement de répliques est utilisée, **PRRW** (Proactive-Reactive Recovery Wormhole), elle est basée sur des rétablissements périodiques (proactifs) et sur des rétablissements déclenchés par

[7] Dans OrBAC, l'expression d'une politique d'autorisation est centrée sur le concept d'organisation, contrairement à RBAC où la politique d'autorisation est centrée sur le concept de rôle [AEK03].

événement (réactifs), afin d'assurer le fonctionnement correcte sans surveillance perpétuelle.
La stratégie proactif-réactive de rétablissement vise à augmenter la fiabilité de CIS et à garantir sa disponibilité, malgré les défauts, les intrusions et les rétablissements.
En particulier, les rétablissements ont des effets bénéfiques (par exemple, les rétablissements réactifs rajeunissent les répliques détectées[8] comme incorrect), mais également des effets négatifs (par exemple, le rétablissement proactif d'une réplique correcte la rend indisponible pendant la durée entière du rétablissement). La propriété principale de la stratégie **PRRW** est que, tant que le défaut montré par la réplique est détectable, cette réplique sera récupérée aussitôt que possible, s'assurant qu'il y a toujours une quantité de répliques disponibles pour soutenir le fonctionnement correcte de cette stratégie [*SOU07*].

Ces mécanismes peuvent être intégrés à une conception basique, non-replié, du CIS d'une manière progressif. Nous pouvons d'abord construire un CIS non-replié, qui fonctionne tout comme un pare-feu, et puis lui ajoutons le mécanisme de rétablissement. Après, une version repliée est établie en utilisant les machines virtuelles dans le même hôte ou dans différentes machines physiques. Ces deux conceptions peuvent plus tard être prolongées pour considérer les répliques additionnelles et pour maintenir la vivacité de système même lorsque quelques répliques sont en état de récupération.

2.5.2 Le mécanisme de tolérance d'intrusion CIS

Dans cette section nous décrivons la conception du CIS intrusion-tolérant, commençant par le raisonnement de conception et avec la définition du modèle de système en présentant la stratégie PRRW, et alors nous montrerons quelques propriétés de service.

2.5.2.1 Raisonnement de conception

Pour comprendre le raisonnement de la conception du CIS intrusion-tolérant, considérez le problème d'implémenter un pare-feu répliqué entre un WAN non-confiant et le LAN confiant que nous voulons protéger. Supposez de plus que nous souhaitons nous assurer que seulement les messages corrects (selon la politique déployée) vont du côté WAN, à travers le CIS, aux ordinateurs stations dans le LAN. Un premier problème est que le trafic doit être reçu par toutes les n répliques, au lieu

[8] Les répliques peuvent employer un détecteur des failles non fiable [*TUS96*] ou un détecteur de silence [*DOU99*].

de 1 seulement (comme dans un pare-feu normal), afin qu'ils puissent exécuter leurs mécanismes actifs de réplication. Un deuxième problème est qu'au plus f répliques peuvent être défectueuses et se comporter avec malveillance, vers d'autres répliques et vers les ordinateurs stations.

La solution du premier problème est d'utiliser un dispositif (par exemple, un Hub Ethernet) pour diffuser le trafic à toutes les répliques. Ceux-ci vérifient si les messages sont conformes à la politique OrBAC et font un vote, approuvant les messages si et seulement si au moins f +1 répliques différentes votent en faveur. Un message approuvé par le CIS est alors expédié à sa destination par une réplique distinguée, le leader, alors il n'y a aucune multiplication inutile du trafic à l'intérieur du LAN.

L'existence des répliques défectueuses est habituellement adressée avec les protocoles masquant du type byzantin, qui extraient le résultat correct à partir des ***n*** répliques en dépit de f défectueux avec malveillance : seulement les messages approuvés par f +1 répliques devraient être transmis (une dont doit être correct car au plus f peuvent être défectueux)

Puisque le résultat doit être envoyé aux ordinateurs stations, l'un ou l'autre il est consolidé à la source, ou à la destination.

L'approche la plus simple et la plus habituelle est d'implémenter un front-end (interface) dans l'hôte de destination qui accepte un message si : (1) f +1 répliques différentes l'envoient ; ou (2) le message a un certificat indique que f +1 répliques l'approuvent [BRA84] ; ou (3) le message a une signature produite par f +1 répliques en utilisant un arrangement de cryptographie de seuil [DES92]. Ceci impliquerait modifier le logiciel des hôtes. Cependant, la modification du logiciel d'un système SCADA/PCS peut être compliquée, et le trafic à l'intérieur du LAN protégé serait multiplié par n dans certains cas (chaque réplique enverrait le message au LAN), ainsi cette solution est indésirable.

Ainsi nous devrions nous tourner à la consolidation à la source, et à envoyer seulement un, mais correct, message. Cependant, ce qui est innovant ici est que les mécanismes de source-consolidation devraient être transparents aux ordinateurs stations (standard). D'ailleurs, une réplique défectueuse (leader ou pas) a accès au LAN (contraire à la proposition de [CHÉ92] où seulement les adaptateurs à panne-silencieuse ont eu accès au LAN) ainsi elle peut envoyer le trafic incorrect aux ordinateurs stations, qui typiquement ne peuvent pas distinguer les répliques défectueuses des répliques correctes. Ceci fait la consolidation à la source un problème dur.

La solution au deuxième problème se trouve en utilisant IPSEC, un ensemble de protocoles standards qui sont attendus être généralisés dans les systèmes SCADA/PCS, conformément aux meilleures recommandations des experts organisationnels et gouvernementaux [*OPP98*]. Dorénavant, nous supposons que le protocole d'en-tête d'authentification IPSEC[9] (Ipsec Authentication Header ''AH'') [*KEN05*] fonctionne dans les ordinateurs stations et dans les répliques CIS. L'idée fondamentale est que les ordinateurs stations accepteront seulement des messages avec un code d'authentification (MAC généré par IPSEC/AH) valide, qui peut seulement être produit si le message est approuvé par f +1 répliques différentes. Cependant, IPSEC/AH MACs sont produits en utilisant une clé partagée K et une fonction d'hachage, ainsi il n'est pas possible d'employer la cryptographie de seuil (*threshold cryptography*).

Comme facette finale de ce modèle, afin d'assurer l'exactitude perpétuelle un mécanisme de rétablissement ''PRRW'' est utilisé, ce dernier évite l'épuisement de ressource dû à la corruption accidentelle ou malveillante des composants [*SOU06*]. L'idée sera expliquée dans la section suivante. Étant donné que le rétablissement proactif peut seulement être mis en application avec des hypothèses synchrones [*SOU05*, *SOU06*], une attache finale est qu'il doit y avoir un certain composant sécurisé et synchrone responsable de déclencher et d'exécuter les rétablissements périodiques. La figure **2.9** décrit l'architecture de CIS.

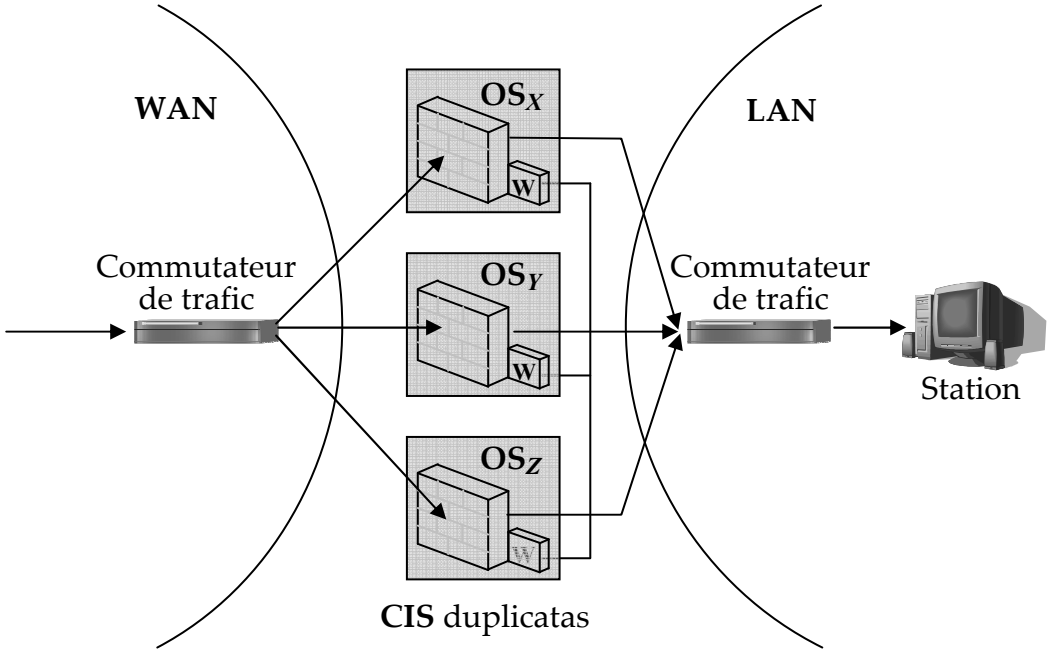


Figure 2.9: L'architecture CIS tolérante aux intrusions

[9] **IPSEC**: **I**nternet **P**rotocol **SEC**urity .

Chaque réplique CIS est déployée dans un système d'exploitation différent (par exemple, Linux, FreeBSD, Windows XP), et les systèmes d'exploitation sont configurés pour employer différents mots de passe et différents pare-feu internes (par exemple, iptables, ipf, pare-feu de Windows). Le *wormhole* (représenté par les petites boîtiers W) est synchrone et assez sécurisé qu'il peut déclencher à l'heure le recouvrement proactif et exécuter un protocole simple de vote qui produit un MAC pour un message si au moins f+1 répliques l'approuvaient (si le *wormhole* est sécurisé, aucune réplique malveillante peut la forcer à signer un message non approuvé).
Un deuxième dispositif de multiplication du trafic (voir la figure, le côté droit) est utilisé pour que les répliques reçoivent tout ce que les autres envoient au LAN. Quand une réplique (correcte) croit qu'une autre réplique est défectueuse (par exemple, si elle envoie des messages non-approuvés au LAN ou c'est le leader et n'expédie pas les messages approuvés), cette réplique est suspectée pour être défectueuse. Quand un quorum des répliques suspecte une réplique, cette dernière sera récupérée.

2.5.2.2 Le Modèle de système

Le système se compose par ***n*** CIS-répliques, CIS = {CIS_1,…, CIS_n }. Ces répliques sont déployées dans l'intersection entre le WAN et le LAN de telle manière que toutes les données traversant les frontières d'un de ces réseaux doivent passer par le CIS. Le modèle de système hybride entoure deux parties [*VER+06*] : le *payload* et le *wormhole.*

■■ Payload

Système asynchrone avec n ≥ af+bk+1 répliques : P_1,…, P_n , sur lequel au plus f répliques peuvent être sujets à des fautes byzantins dans une période donnée (i.e. de rétablissement), et au plus k répliques peuvent être récupérés en même temps. Si une réplique n'échoue pas entre deux rétablissements elle serait correcte, autrement elle serait défectueuse. Nous assumons l'indépendance de défaut pour les répliques, c.-à-d., la probabilité qu'une réplique soit compromise est indépendant à la défaillance d'une autre réplique. Cette hypothèse peut être justifiée dans la pratique en utilisant plusieurs types de diversité (i.e. le rétablissement d'une certaine réplique s'effectue en modifiant les vulnérabilités qui peuvent être exploitées par un ennemi malveillant, par exemple, les mots de passe d'accès d'OS, les ports communicatifs, etc.) [*OBE06*].

Finalement, le payload exécute les applications et les protocoles « normales » et nous supposons également que les ordinateurs stations ne peuvent pas être compromis[10].

■■ Wormhole

Sous-système synchrone $W = \{W_1,\ldots, W_n\}$ dans lequel au plus $fc \leq f$ wormholes locaux peuvent faillir par accident. Nous supposons que quand un wormhole local W_i s'écrase, le payload correspondant CIS_i s'écrase aussi. Le canal de contrôle employé par le wormhole est isolé et synchrone, offrant une sémantique de multicast fiable. Chaque wormhole local et les ordinateurs stations partagent une clé symétrique K, les ordinateurs stations acceptent seulement les messages authentifiés en utilisant cette clé. Ce wormhole peut être déployé comme ensemble de composants dignes de confiance (un par réplique) reliés par un canal séparé de contrôle. La figure **2.10** illustre ce modèle.

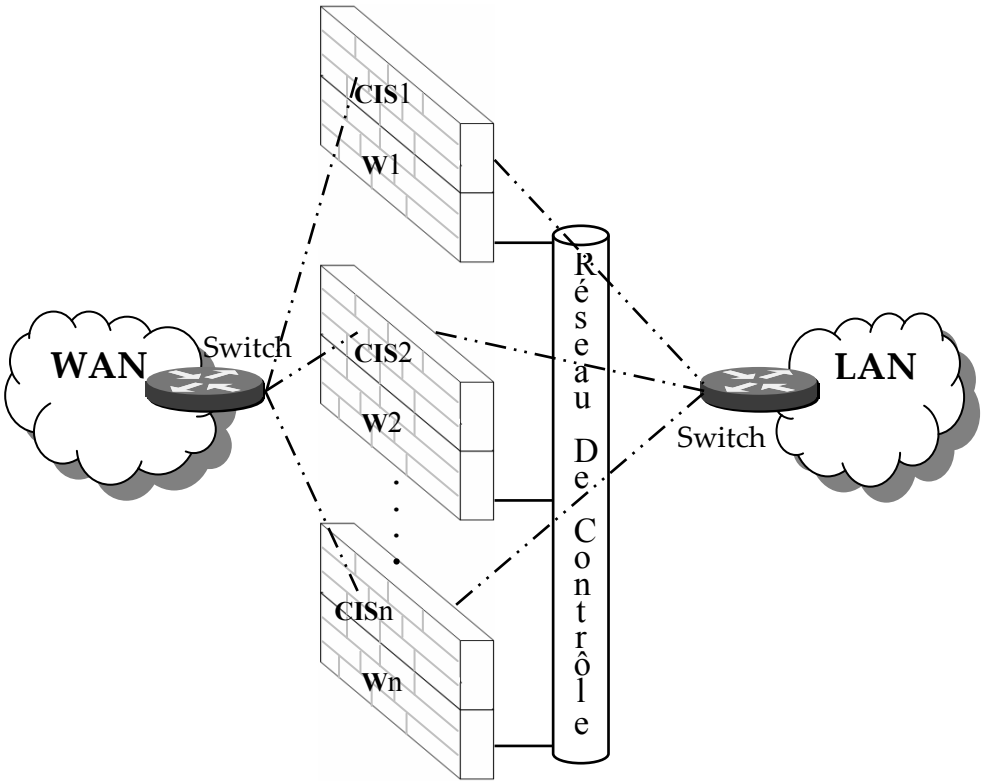


Figure 2.10: Le modèle de système

[10] C'est le réseau de confiance que nous visons à protéger. D'ailleurs, il n'a pas de sens de protéger un hôte qui peut être compromise et attaque le pare-feu.

■■ La stratégie PRRW

Nous expliquons maintenant la stratégie **PRRW** (Proactive-Reactive Recovery Wormhole) [*SOU07*]; ce service a besoins des informations fournies par les payloads afin de déclencher des rétablissements réactifs. Cette information est obtenue à travers deux fonctions d'interface : W_suspect(j)[11] et W_detect(j)[12]. La stratégie PRRW gère les récupérations des répliques en utilisant un mélange des rétablissements proactifs et réactifs, et elle est caractérisée par les paramètres suivants :

– L'intervalle maximum de temps T_P (cycle ou période de rétablissement) entre les rétablissements consécutifs sur la même réplique (par conséquent chaque réplique est récupérée au plus après T_P).
– Le temps d'exécution d'un rétablissement est T_D (le pire des cas).
– Le nombre maximum ***k*** des répliques qui peuvent récupérer simultanément.
– ***f*** est le nombre maximum des répliques simultanément corrompues et que le système peut les tolérer.

La stratégie **PRRW** est organisée suivant les indications de la figure **2.11** : le temps est divisé en $\lceil n/k \rceil$ différentes slots de temps qui sont cycliquement répétées. Chaque slot est divisé en deux tâches : la tâche A et la tâche R_i, avec i =1,…, $\lceil n/k \rceil$.
Les rétablissements proactifs (périodiques) sont exécutés pendant la tâche Ri seulement ; au plus ***k*** répliques peuvent être récupérée simultanément dans chaque tâche Ri, selon l'index de réplique.
La réplique ***i***, avec i =1,…, k, sont récupérées dans la tâche R_1, la réplique i, avec i = k+ 1,…, 2k, sont récupérées dans la tâche R_2 et ainsi de suite. La tâche R_i dure (au plus) T_D et elle est exécutée encore après une période T_P.
Deux types de rétablissements réactifs (apériodiques) peuvent être déclenchés sur une réplique ***i*** :

1. Rétablissement réactif « immédiat », déclenché si un quorum de f+1 accusations existe concernant i qui a envoyé des messages illégaux ; en ce scénario la réplique ***i*** est « détectée » d'être compromis, parce qu'au moins une réplique correcte a détecté qu'elle est échouée.

[11] Un payload i appelle W_suspect(j) pour informer le PRRW que la réplique j est suspectée d'être échouée.
[12] Si la réplique i sait sans doute que la réplique j est échouée, alors W_detect (j) est appelée à la place.

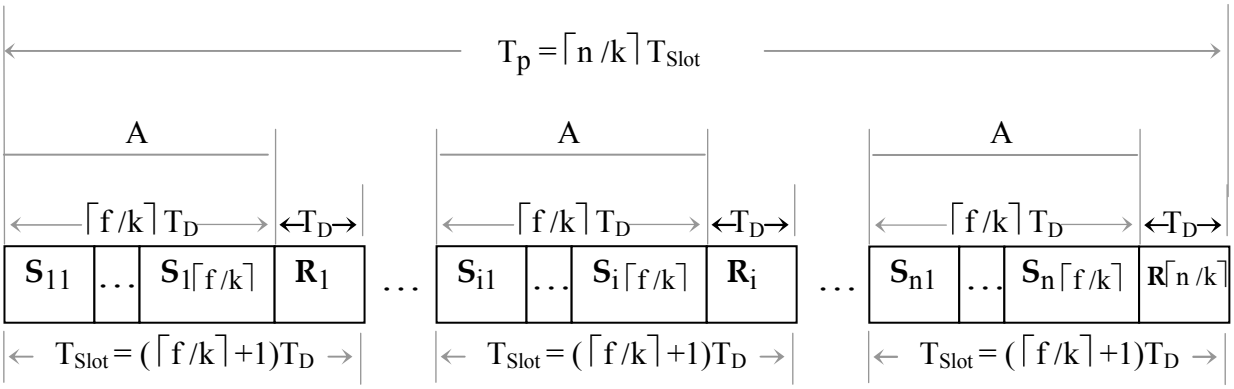


Figure 2.11: L'ordonnancement des tâches de PRRW (*rejuvenation process*)

2. Rétablissement réactif « retardé », déclenché si un quorum d'au moins f+ 1 accusations existe au sujet du leader courant i, parmi lesquelles i envoie des messages illégaux, autres i n'expédie pas un message signé (les messages signés n'ont pas été expédiés pendant un temps O_t). Dans ce scénario le leader i est « suspecté » d'être compromis, parce qu'au moins une réplique correcte a soulevé une accusation au sujet de la réplique leader ***i***.

Les rétablissements réactifs « immédiats » sont immédiatement déclenchés sur la réplique ***i*** dès qu'elle est détectée d'être compromis.

Les rétablissements réactifs « retardés » sont seulement déclenchés sur la réplique leader, sont exécutés pendant la tâche A et sont coordonnés avec des rétablissements proactifs.

Si aucun rétablissement réactif « immédiat » n'est déjà déclenché pour la réplique ***i***, la stratégie PRRW trouve le sub-slot de rétablissement le plus étroite où le rétablissement de la réplique i ne met pas en danger la disponibilité du CIS. Si le sub-slot trouvé est situé dans le slot où la réplique ***i*** sera proactivement récupérée, le rétablissement réactif « retardé » n'est pas exécuté. La tâche A est divisé en $\lceil f/k \rceil$ sub-slots de rétablissement identifiées comme Sij; jusqu'à k répliques peuvent être récupérées simultanément dans chaque sub-slot. La tâche A dure (au plus) $\lceil f/k \rceil \times T_D$.

Chaque slot dure par conséquent jusqu'à $(\lceil f/k \rceil + 1)\, T_D$ avec la période T_P. Après que chaque tâche Ri ait été exécutée une fois, chaque réplique a été proactivement récupérée une fois.

Un nouveau leader est élu par le wormhole si le leader courant est en train de se récupérer ou si le wormhole local du leader courant est détecté d'être écrasé. Le nouveau leader est choisi comme réplique (actuellement non écrasé) la plus récemment récupéré par un rétablissement proactif.

■■ **Réseaux**

En outre le modèle de répliques CIS, nous devons définir les hypothèses fondamentales de la communication entre LAN et WAN. Nous considérons que les messages qui arrivent aux répliques CIS du WAN et du LAN ont une sémantique juste non fiable de multicast, une adaptation de l'abstraction juste de liens au multicast : si un message est diffusé infiniment autan de fois qu'il sera reçu par tous ses récepteurs infiniment autan de fois. Les deux primitifs offerts par ce service sont : U-multicast(G, m), pour diffuser un message m au groupe G, et U-receive(G, m), pour recevoir m qui était diffusé au G. Ceci est établi dans la pratique par les dispositifs de réplication du trafic. Nous supposons que toute la communication entre les répliques et d'autres machines du WAN et du LAN sont basées sur ces primitifs. Pour le LAN, nous supposons également que la communication est source-authentifiée et l'expéditeur d'un message m est dénoté par sender(m).

■■ **Cryptographie**

On assume que les MACs utilisés dans IPSEC/AH héritent de la propriété de résistance de collision des fonctions de hachage en lesquelles ils sont basés [*OPP98*], c.-à-d., qu'il est infaisable de trouver deux messages que pour une clé K ayant le même MAC.

2.5.2.3 Les propriétés de service

Tout d'abord, laissez-nous définissent le concept d'un message légal : un message serait légal s'il est conforme à la politique déployée courante ***P***. un message qui n'est pas conforme avec P serait illégal. D'ailleurs, un message serait traité par sa machine de destination si son contenu est fourni à la couche application (par exemple, le système SCADA). Les propriétés de base offertes par le CIS sont les suivantes :

■■ **Validité :** Un message légal reçu par au moins une réplique correcte (qu'elle n'est pas récupérée pour assez longtemps) est expédié à sa destination ;

■■ **Intégrité :** Un message illégal n'est jamais traité par sa machine de destination.

Notez que ces deux propriétés sont suffisamment faibles pour être satisfaites par un système de communication multicast juste et non fiable et assez fort pour s'assurer que seulement des messages légaux seront traités aux hôtes de LAN.

2.6 Conclusion

Les protocoles tolérants aux Intrusions sont suffisants pour assurer la résilience des systèmes repliés tant que pas plus qu'un certain nombre fini de répliques ne sont compromis. Malheureusement, cette garantie a l'utilité limitée dans les systèmes de longue vie exposés aux attaques malveillantes. Ce chapitre a présenté le mécanisme CIS, un service qui peut être employé pour augmenter la résilience des systèmes repliés intrusion-tolérants fonctionnant dans les environnements malveillants afin de protéger les infrastructures critiques (i.e. SCADA, PCS ….). L'intrusion-tolérant CIS enlève les défauts et les intrusions qui peuvent se produire pendant l'exécution de système, et un tel enlèvement est fait en combinant les techniques de sécurité et de rétablissement.

Cependant, la tolérance aux intrusions seule ne répond pas aux exigences de la sécurité dans les systèmes distribués et décentralisés tels que les infrastructures industrielles sensibles aux attaques malveillantes. En effet, ce mécanisme protège la globalité de l'infrastructure d'être échouée mais il n'effectue aucun contrôle sur les informations après leur distribution.

Le mécanisme de protection CIS présenté dans la section précédente sera plus complet et assure une sécurité fiable pour les infrastructures critiques (ICs) s'il est étendu pour effectuer le contrôle de flux d'information sur les communications entre les objets du CIS.

Dans le prochain chapitre, nous allons présenter la notion de contrôle de flux d'information et un modèle basé sur l'analyse du code qui sera intégré au modèle CIS pour obtenir une approche hybride décrite dans le chapitre **4**.

CHAPITRE

3

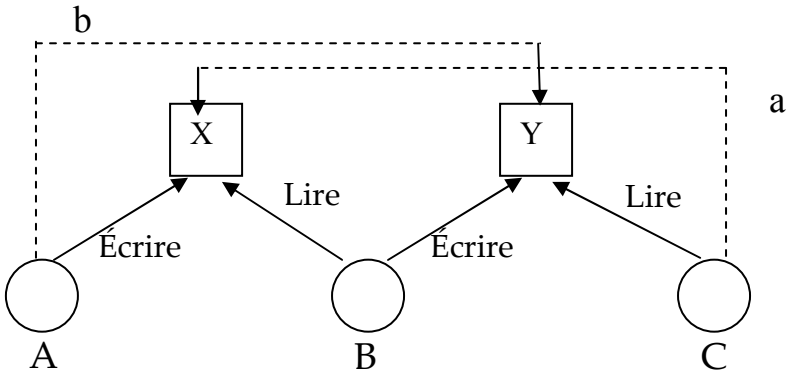
Contrôle de Flux d'Information

3.1 Introduction

Une des tendances les plus remarquables dans le calcul moderne est la prolifération des systèmes automatisés aux milieux commerciaux, administratifs et autres champs. La quantité d'information sensible étant stockée dans les systèmes informatiques a augmenté, par conséquent et avec cette réalité, le sérieux d'une attaque (interne ou externe) sur un système où un attaquant réussit à accéder illégalement aux données dans le système. Les systèmes distribués ont aggravé ce problème puisque l'information peut être accédée à partir des distances géographiques larges et non seulement l'existence de plusieurs moyens d'attaque, mais aussi il est plus difficile de contrôler l'activité de traitement d'un système qui mène à ces attaques sur la sécurité, ou sur la confidentialité des informations stockées.

Les systèmes informatiques utilisent typiquement le mécanisme de contrôle d'accès pour assurer la confidentialité et l'intégrité des informations. Le principe basique des modèles de contrôle d'accès est d'associer une clé d'accès à chaque opération sur un objet, uniquement le processus possédant une clé peut exécuter

l'opération associée sur un objet [*BRY95*]. L'exemple de la figure **3.1** montre que la confidentialité d'information (a) et l'intégrité d'information (b) qui est un problème dual à la confidentialité ne peuvent pas être assurées par le contrôle d'accès seul :



a : lecture interdite
b : écriture interdite

Figure 3.1 : Un flux d'information non sécurisé

Dans l'exemple, l'utilisateur **B** peut lire l'information de l'objet **X** puis l'écrire dans l'objet **Y.** L'utilisateur **C** a le droit de lire l'objet **Y** et ainsi il accède indirectement à l'information contenue dans l'objet **X** même s'il n'est pas autorisé à lire **X**, ce qui induit à une divulgation d'information. De la même manière, l'utilisateur **A** écrit une information dans l'objet **X** qui sera écrite dans **Y** par l'utilisateur **B**. Par conséquent, l'utilisateur **A** peut écrire indirectement dans **Y** même s'il n'est pas autorisé à écrire **Y**, ce qui induit à une violation d'intégrité de l'objet **Y**.
Un mécanisme de contrôle des informations après leur dissémination est donc nécessaire pour compléter la tâche du contrôle d'accès dans les systèmes manipulant des informations sensibles afin d'assurer une meilleur protection de ces dernières.

Dans ce chapitre, nous introduisons la notion de contrôle de flux d'information, nous résumons la taxonomie des modèles de contrôle de flux d'information, nous citons des travaux significatifs dans différents domaines et nous présentons un exemple de modèle de contrôle de flux d'information qui servira d'introduction à notre approche de contrôle de flux pour les systèmes d'infrastructures critiques.

3.2 Le contrôle de flux d'information

Le contrôle de flux d'information (**IFC** : **I**nformation **F**low **C**ontrol) est un aspect de la sécurité informatique qui consiste à régulariser le flux d'information entre les objets d'un système informatique pour assurer la confidentialité et l'intégrité des informations.

3.2.1 Définitions

La transmission d'information entre les processus se distingue de flux d'information conséquent qui peut être imposé entre les objets [*BER98*].

■■ La transmission d'information

Chaque message entre les objets a comme conséquent une transmission d'information, la transmission *vers l'avant* diffuse l'information de l'expéditeur au récepteur par la liste de paramètres de message. La transmission *en arrière* diffuse l'information du récepteur à l'expéditeur par la réponse. La transmission d'information peut être directe par l'envoi de messages entre deux processus ou indirecte sans échange de messages. Par exemple, si un processus P_i invoque un processus P_j qui à son tour invoque un processus P_k alors il y a une transmission directe à travers les paramètres du message, de P_i à P_j et de P_j à P_k, et par conséquent, une transmission indirecte de P_i à P_k.

On appelle un *canal* une transmission directe ou indirecte d'information. Selon la direction de l'information transmise, on distingue entre un canal *vers l'avant* (de l'émetteur au récepteur) et un canal *en arrière* (du récepteur à l'émetteur).

■■ Le flux d'information (FI)

Il existe un flux d'information entre un objet o_h et un objet o_k quand l'information sur l'état de o_h est rangée dans o_k. Un flux d'information intuitivement exige une transmission d'information entre quelques processus fonctionnant sur les deux objets. Cependant, la transmission d'information entre des processus fonctionnant sur différents objets ne provoque pas nécessairement un flux. Nous considérons ces deux suppositions [*BER98*] :

i) Un flux d'information à partir d'un objet peut exister seulement si l'information est *lue* à partir de cet objet.

ii) Un flux d'information vers un objet peut exister seulement si l'information est *écrite* dans cet objet.

Un flux d'information d'un objet o_h vers un objet o_k qui est dénoté par $o_h \Rightarrow o_k$ nécessite :

1) Une opération de *lecture* à partir de l'objet ''source'' o_h,
2) Une opération d'*écriture* ou de *création* sur l'objet ''destinataire'' o_k et
3) Un *canal de transmission* du processus de lecture vers le processus d'écriture.

■■ Un flux d'information sain

Un flux d'information sain (sûr) est un flux qui ne provoque pas une fuite d'information vers les utilisateurs non autorisés à y accéder. Un flux d'information d'un objet obéit aux spécifications de sécurité (c.-à-d. les accès qui sont ou qui ne sont pas permis) si et seulement si les utilisateurs qui sont autorisés à lire un objet :

1) Sont permis de lire l'information coulée vers eux soit par une autorisation explicite dans la liste de contrôle d'accès (ACL) soit par des exceptions laxistes durant la transmission d'information,
2) Ne sont pas empêchés de lire cette information par des exceptions restrictives (négatives) appliquées durant la transmission d'information.

En d'autres termes, le flux d'information est sain si tous les utilisateurs autorisés à lire un objet sont autorisés de voir l'information transmise vers cet objet (selon l'ACL de l'objet duquel l'information a été lue et les exceptions produites le long de la transmission).

L'exemple de la figure **3.2** résume le problème de la sécurité durant les flux d'informations. Un observateur possède une copie du code d'une application. L'observateur peut voir ce qui se passe pendant l'exécution car la politique de sécurité du système lui permet de lire les valeurs de certaines variables. Dans la figure, les variables que l'observateur est autorisé de voir sont désignées par les disques entourés par le cercle, les disques en dehors du cercle représentent les variables contenant les informations que l'observateur est interdit de les voir. La condition de sécurité est la suivante : étant donné que l'observateur connaît le texte du programme du système, il ne doit pas être capable de déduire les valeurs des variables en dehors de cercle via les valeurs des variables à l'intérieur de ce cercle.

En d'autres termes, il ne doit pas y avoir un flux d'information des variables en dehors du cercle vers celles de l'intérieur :

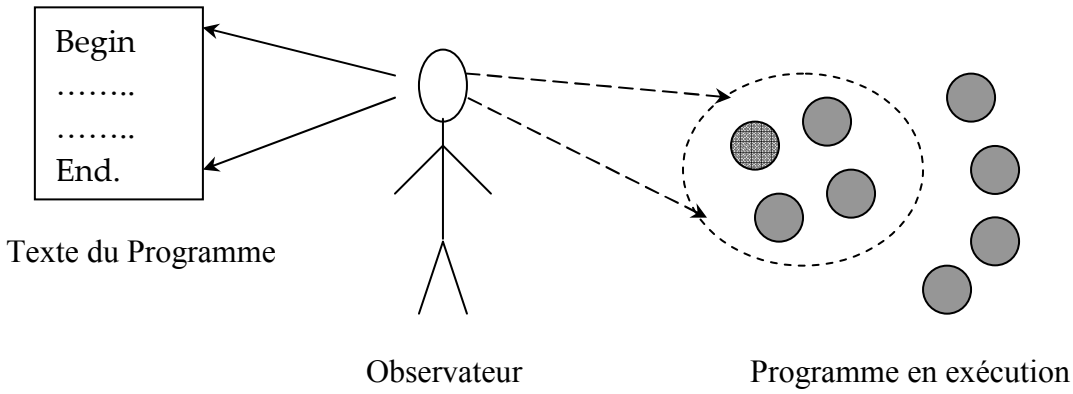


Figure 3.2 : Exemple de contrôle de flux d'information [*BRY95*]

3.2..2 Les concepts de contrôle de flux d'information

Le contrôle de flux d'information (CFI) introduit de nouveaux concepts que nous essayons de définir dans cette section.

■■ Un flux d'information direct et indirect

Un flux d'information direct est causé par une influence explicite d'un objet sur un autre. Un flux direct d'un objet a à un objet b se produit en assignant la valeur de a à b : $(a \leftarrow b)$. Le flux d'information indirect est causé par une influence logique d'un objet sur un autre. Par exemple, la figure **3.3** illustre un flux d'information causé par une structure de contrôle d'un programme, où la valeur secrète de qc peut être déterminée en examinant la valeur de la variable publique qa après l'exécution du programme. Ainsi, il y a un flux implicite de qc vers qa :

```
Public qa;
Secret qc;
qa = false;
If (qc == true)
{
qa = true;
}
```

Figure 3.3 : Exemple de flux d'information indirect [*TAR06*]

■■ Le contrôle de flux d'information statique et dynamique

Le contrôle de flux d'information statique consiste à définir (à travers des règles de flux d'information) et éliminer les flux d'information illégaux avant l'exécution du système sécurisé (à la compilation par exemple). Dans le contrôle de flux d'information dynamique, les flux illégaux sont découverts avant et durant l'exécution.

Le contrôle de flux d'information statique ne consomme pas du temps d'exécution (minimise l'*overhead*) et permet de discerner les flux implicites. En revanche, il a une vue partielle du système réel. Le contrôle de flux dynamique a une vue plus grande sur le système, ces deux techniques peuvent être combinés.

■■La flexibilité d'un contrôle de flux d'information

Le contrôle de flux d'information impose une restriction sur les systèmes contrôlés puisque parfois il interdit quelques flux légaux. La flexibilité vue dire que tous les flux d'information légaux sont autorisés. Ainsi, elle peut être réalisée en définissant des exceptions aux règles de contrôle de flux d'information d'une manière à préserver l'exécution normale du système. Par exemple dans un cas d'alerte, on peut autoriser le flux du mot de passe d'un objet à un autre afin d'obtenir l'accès à quelques ressources bien que ce n'est pas permis.

3.3 La taxonomie des modèles de CFI

Généralement, on peut catégoriser les travaux de contrôle de flux d'information en trois classes : le contrôle de flux basé sur les langages de programmation, le contrôle de flux basé sur les messages et le contrôle de flux basé sur la théorie de l'information [*BER06*].

3.3.1 Le CFI basé sur les langages de programmation

Le contrôle de flux d'information basé sur les langages de programmation consiste à utiliser un système de type de sécurité. Des annotations sont associées aux types de variables et expressions d'un programme pour spécifier des politiques sur l'utilisation des données de ces types. Ces politiques de sécurité sont ensuite appliquées par la vérification de type à la compilation. Ainsi, ce contrôle ne consomme pas du temps d'exécution.

Les mécanismes basés sur les langages sont utilisés pour différents objectifs de la sécurité. Le mécanisme de sécurité de l'environnement Java est un exemple bien

connu y compris le vérificateur de *byte code* [*LIN99*], le modèle sandbox [*SUN07*] et l'inspection de pile (*the stack inspection*) [*WAL00*]. Tous ces mécanismes sont basés sur le langage, en d'autres termes, ils sont appliqués via le langage Java.
Le travail de Denning pour l'analyse de flux sécurisé et le modèle de treillis [*GRA72*] a été le premier qui a initié la certification de programme, qui est une forme efficace d'analyse statique et qui a pu être facilement incorporée dans un compilateur pour vérifier les flux d'informations sécurisés dans les programmes.
D'autres travaux et modèle de contrôle de flux d'information pour les langages de programmation sont cités dans la section **3.4**.

Le contrôle de flux d'information basé sur les langages de programmation présente différents inconvénients. La stratégie de contrôle d'information doit être fixée à la compilation par le développeur de l'application et ne peut pas être modifiée pendant l'exécution. En outre, l'intégration de contrôle de flux d'information dans les langages existants nécessite une compréhension profonde des compilateurs des langages de programmation qui n'est pas toujours une solution pratique. De plus, ce type de contrôle n'est pas convenable aux systèmes distribués et hétérogènes où les applications sont écrites dans différents langages de programmation et s'exécutent sur des plateformes incompatibles. Pour ces raisons, le contrôle de flux d'information basé sur les messages est apparu.

3.3.2 Le CFI basé sur les messages

Dans ces systèmes, les composantes peuvent communiquer uniquement par l'envoi de massages. Le mécanisme de contrôle de flux d'information intercepte les messages passés entre les composantes du système et applique les politiques de sécurité pour permettre ou interdire ces flux.
McCollumn et al [*MCC90*] ont proposé une nouvelle forme de contrôle d'accès qui impose des restrictions sur le flux d'information dans le système. Leur modèle permet à un propriétaire d'un fichier d'associer à chaque objet ***o*** contenu dans ce fichier une liste de contrôle d'accès (ACL). Cette liste se propage via les labels des sujets et des objets à tous les objets que ***o*** peut affecter.
Une approche similaire a été proposée par Stoughton [*STO81*]. Dans cette proposition, chaque objet a deux attributs de protection : l'attribut d'accès *courant* décrit ce qui doit être fait par le destinataire de l'information de cet objet ; et l'attribut d'accès potentiel décrit ce qui doit être fait avec l'information même si elle est copiée dans un autre objet.

Boebert et Furguson [*BOE85*] ont proposé de contrôler les *Trojan Horse*[1] en interposant un sous système protégé entre le programme suspecté et le système de fichiers. Une autre approche proposée par Karger [*KAR87*] contrôle les effets des *Trojan Horse* dans les systèmes discrétionnaires en limitant les fichiers accessibles par les programmes d'application à base de quelques connaissances sur les programmes eux-mêmes. Tous les accès demandés par les applications sont arbitrés par un contrôleur de messages. Le contrôleur de message compare le nom de l'objet à accéder au modèle spécifié pour l'application. Si le nom de l'objet satisfait le modèle alors l'accès est accordé. Elisa Bertino, Sushil Jajodia et al. [*BER97*] ont décrit un modèle de contrôle de flux d'information d'une grande assurance pour les systèmes Orienté Objet. Dans ce modèle, les flux d'information bidirectionnels sont produits par l'envoi de messages où un filtre de messages intercepte chaque message échangé entre les objets pendant l'exécution d'une transaction pour garantir qu'aucun flux illégal ne se produit.

Les travaux dans ce domaine sont motivés par la popularisation des systèmes d'appel de procédures distantes (**RPC** : **R**emote **P**rocedure **C**all), les systèmes distribués Orientés Objet (tel que *CORBA*) et les Services Web. Néanmoins, la supposition que les informations circulent uniquement par l'échange de messages les rend inappropriés pour certains systèmes où l'information peut circuler par les relations d'hiérarchie et d'agrégation entre les objets.

3.3.3 Le CFI basé sur la théorie de l'information

La plupart des travaux dans le contrôle de flux d'information sont intéressés par les flux causés par les transmissions au sein des réseaux et la sauvegarde d'information. Cependant, d'autres types de flux d'information peuvent mener à des fuites d'information. Les flux d'information peuvent être interprétés par les modèles de la théorie de l'information. Ceci est principalement dû au fait que les systèmes non déterministes peuvent montrer des canaux secrets probabilistes (*probabilistic covert channels*)[2] qui ne sont pas régnés par les modèles standard de la sécurité informatique. McLean [*MCL90*] a donné un traitement très général des modèles de sécurité de flux d'information.

---

[1] Le Trojan Horse est un programme dans lequel une partie nocive (par exemple, le sous-programme qui cause la fuite de l'information aux lecteurs non autorisés) est contenue à l'intérieur et semble un code inoffensif.

[2] Les mécanismes de transmission d'information à travers les systèmes informatiques sont appelés canaux. Les canaux qui exploitent un mécanisme dont le but primaire n'est pas le transfert de l'information s'appellent les canaux secrets.

Sa définition de sécurité (appelée FM) est donnée sous forme d'équation entraînant des probabilités conditionnelles. J.Gray [*GRA91*] a proposé un modèle général et probabiliste de machine à état qui peut être utilisé pour modéliser une grande classe des systèmes informatiques non déterministes et déterministes. Il a développé la théorie des probabilités nécessaire pour énoncer et montrer des propriétés de flux d'information du système modélisé.

Tanis que le contrôle de flux d'information basé sur la théorie de l'information offre une assurance élevée concernant la non divulgation d'information à travers les canaux secrets, son manque à une implémentation concrète demeure une question à résoudre.

3.4 Exemples de techniques de CFI

Dans ce qui suit, nous résumons quelques techniques récentes de contrôle de flux d'information :

Un système de type puissant pour une analyse de flux sécurisé a été proposé par Volpano [*VOL96*]. Le modèle contribue au contrôle de flux d'information des programmes dans le contexte des nivaux de sensibilité multiples. En relation avec les sémantiques des langages de programmation, la vérification de la sécurité est confirmée en montrant que les programmes bien typés sont caractérisés par la propriété de non interférence[3].

Une approche basée sur les travaux de Denning [*DEN76*] pour l'analyse de flux d'information des programmes a été proposé par Bryce, Banâtre & Le Métayer [*BRY95*]. Le problème de base adressé par cette approche est que la plupart des systèmes emploient les logiciels du domaine public et que les violations de sécurité peuvent se produire non seulement par la lecture illégale de l'information, mais aussi par l'inférence des informations confidentielles à partir d'autres informations en utilisant la connaissance de code du programme, l'approche adoptée pour aborder ce problème est basée sur l'analyse du code, où la sémantique du flux d'information pour un langage de programmation parallèle est définie, et un système de preuve axiomatique est développé à partir de cette sémantique. Ce système de preuve permet à la sécurité des systèmes parallèles d'être vérifiée en validant des attributs sur les flux d'information entre les variables qui sont insérées dans le code du programme.

[3] Les données confidentielles ne doivent pas interférées avec (ou affecter) les données publiques visibles par d'autres utilisateurs. Une politique de ce genre est appelée la politique de non interférence.

Le travail le plus récent dans le domaine de contrôle de flux d'information pour les langages de programmation est celui de Myers et Liskov [*MYE97*]. Ils ont proposé un modèle de labels avec une administration décentralisée. Le modèle permet aux utilisateurs de contrôler le flux de leurs informations d'une manière décentralisée et sans imposer des contraintes rigides d'un système de sécurité à multi niveaux traditionnel. En outre, ils ont montré que la vérification statique des labels peut être utilisée pour certifier la légalité de flux d'information tout en réduisant la vérification dynamique des labels. Par la suite, Myers et Liskov ont montré dans [*MYE00*] comment le modèle décentralisé de labels peut être utilisé pour protéger l'intimité (*privacy*). De plus, ils ont introduit une extension au langage Java (appelée **Jif** : **J**ava **I**nformation **F**low) qui fournit un mécanisme de vérification statique en utilisant le modèle de labels proposé.

Un autre travail de Myers [*MYE99*] a introduit un nouveau langage, appelé Jflow, pour la vérification statique de flux d'information. C'est une extension au langage Java en ajoutant des annotations aux informations statiquement vérifiées. Le langage proposé inclut le modèle de labels décentralisé, le polymorphisme de labels, la vérification dynamique de labels et l'inférence automatique de labels.

Un travail récent de B.Li [*BLI02*] a analysé le flux d'information dans les programmes Java et a proposé une technique de découpage en tranches. L'utilisation de cette technique permet le calcul de la quantité, de la largeur et de la corrélation de flux d'information.

Une approche d'analyse de flux d'information pour un système *RBAC* est proposée dans [*OSB02*]. Dans cette approche, à partir d'un graphe arbitraire de rôles est produit un graphe de flux d'information. D'une manière semblable, Belokosztolszki et al. [*BEL03*] ont présenté une approche qui examine les fuites d'information involontaires pendant l'application des politiques de modèle *RBAC*[4].

Une adaptation du modèle de Myers et Liskov a été proposé par Z.Tari & al. [*TAR06*]. Cette adaptation consiste à protéger la confidentialité des données manipulées par un système de Services Web en empêchant la divulgation d'information, ce modèle utilise un mécanisme du contrôle de flux d'information basé sur la vérification dynamique des labels des objets manipulés par les Services Web.

[4] Le modèle *RBAC* (*R*ole *B*ased *A*ccess *C*ontrol) propose de structurer l'expression de la politique d'autorisation autour du concept de rôle. Un rôle est un concept organisationnel : des rôles sont affectés aux utilisateurs conformément à la fonction attribuée à ces utilisateurs dans l'organisation.

3.5 Une approche basée sur l'analyse du code de Bryce

Dans cette section, nous décrivons le modèle de contrôle de flux d'information introduit par Bryce, Banâtre & Le Métayer [*BRY95*].

Comme nous avons mentionné le problème de base adressé par cette approche est que la violation de sécurité peut se produire non seulement par la lecture illégale de l'information, mais aussi par l'inférence des informations confidentielles à partir d'autres informations en analysant le code du programme. Pour illustrer ce problème, nous reprenons l'exemple de la figure **3.2**, laissez x être une variable extérieur du cercle, et que la politique de sécurité du système interdit l'observateur de savoir sa valeur; laissez y être une variable à l'intérieur du cercle. Si un programme contenant l'affectation simple : y ← x est exécuté, alors un flux d'information illégal se produit au domaine de l'observateur : l'observateur peut déduire la valeur de x de la valeur du Y [*BRY95*]. Pour attaquer ce problème cette approche se base sur un langage de programmation parallèle standard, ils ont choisi **CSP** (**C**ommunicating **S**equential **P**rocesses) [*HOA78*], et pour définir l'ensemble de flux d'information que chaque commande du langage s'effectue. En formalisant la description de ces derniers, la sémantique de flux d'information du langage est définie.

Un programme CSP contient un nombre fixe de processus, chacun identifié par une chaîne de caractères 'label'. Ce 'label' peut également être un tableau où l'entrée label[i] appelle un processus qui a le même code que les autres processus appelés dans le tableau. Les processus se communiquent par *rendez-vous* bidirectionnel : chaque processus appelle le processus avec lequel il veut communiquer ; quand une partie (processus) dans la communication exécute sa commande de communication, il se bloque jusqu'à ce que le processus partenaire soit prêt à exécuter sa communication. L'effet de la communication est d'assigner la valeur de l'expression évaluée dans le processus d'envoi à la variable du processus de réception (appelé dans la commande de réception).

3.5.1 La sémantique de flux d'information du langage

Pour raisonner aux flux d'information d'un programme, une certaine manière de représenter ces flux est nécessaire, on a besoin notamment d'une certaine manière de représenter l'ensemble de variables, l'information au sujet derrière a coulé vers (i.e.

condition vérifiée), ou a influencé, une variable v. Ils ont appelé cet ensemble la variable de sécurité de v, dénoté $\overline{v}$.
Les flux indirects sont modélisés par la variable *''indirect''*, dont il ya une par processus, et est défini comme une séquence d'ensembles de variables :

$$indirect : (\mathbf{P}variables)^{+}$$

Où P est '' l'ensemble des opérateurs '' et + pour dire que l'ensemble n'est pas vide, pour un ensemble vide ou nul est $\prec \{\} \succ$. La valeur de flux pour *indirect*, dénoté *val(indirect)* est l'ensemble de toutes les variables dedans la variable *indirect*.

Soit *indirect(i)* dénote la $i^{ème}$ entrée :

$$val(indirect) \triangleq \bigcup_{i=0}^{n-1} indirect(i)$$

Où $\cup$ est l'opérateur d'union et car indirect est une séquence (d'ensembles), on suppose les opérateurs : *head(), tail()* et concaténation (o). Un ensemble de variables V peut être ajouté à, par opposition à concaténer avec, *indirect*. Ceci est fait avec l'opérateur $\uplus$.

$$V \uplus indirect \triangleq \mathrm{o}_{i=0}^{n-1} (V \cup indirect(i))$$

Finalement, ils ont utilisé l'opérateur $\sqcup$ pour combiner deux variables *''indirect''* ensemble, son effet est d'ajouter – en utilisant l'opérateur $\uplus$ – les variables des deux *indirects* dans les entrées de la première *indirect*.

$$indirect_i \sqcup indirect_j \triangleq (val(indirect_i) \cup val(indirect_j)) \uplus indirect_i$$

Ils ont défini l'*état de sécurité du flux* d'un programme comme suit, premièrement, associer chaque variable à sa sécurité variable et deuxièmement la valeur de *indirect*.
En décrivant le comportement des commandes avec le respect de l'état de sécurité du flux ils ont utilisé une notation opérationnelle : la sémantique du flux est définie comme une relation de transition '' $\rightarrow$ '' qui associée un couple programme::état à un autre couple. Par exemple l'interprétation de $(C_1, \sigma) \rightarrow (C_2, \tau)$ est que l'exécution de la commande C_1 sur l'état de sécurité du flux σ mène à un autre état τ sur lequel la commande C_2 s'exécute.

3.5.2 La sémantique de flux de la commande d'affectation

Soit la commande $y := \exp(x_1, \ldots\ldots, x_N)$, cette commande affecte la variable de sécurité $\overline{y}$ à :

$$\{x_i \mid i = 1..N\} \cup \{\overline{x_i} \mid i = 1..N\} \cup val(indirect)$$

Où le terme $\{x_i \mid i = 1..N\}$ capture le flux d'information direct à partir de chaque variable x_i. Le terme $\{\overline{x_i} \mid i = 1..N\}$ capture la transitivité dans les flux d'information. Par exemple, le programme [$b := a$; $c := b$] cause un flux de la variable a et b vers la variable c puisque c contient la valeur de a et de b [5]. Comme a été mentionné $val(indirect)$ sauvegarde la valeur du flux que la variable de côté gauche recevra indirectement. Les flux indirects sont examinés bientôt dans le contexte des commandes réitérées et alternatives.

Un résultat de cette approche est qu'une variable x peut être un membre de la variable de sécurité $\overline{y}$ quoique y ne puisse pas être employé pour déduire la valeur courante de x ; une tâche suivante a pu changer x. Tant que la valeur courante de y dépend fonctionnellement d'une valeur (peut-être ancien) de x, alors x sera dans $\overline{y}$.

Bien sûre, quand y est remis à zéro ou prend une affectation n'impliquant pas x, alors x n'est plus un membre de la variable de sécurité $\overline{y}$ puisque le nouveau contenu de l'information du y est indépendant de n'importe quelle valeur précédente de x. Une approche alternative, où des instances des variables dans le programme sont étiquetés est décrite dans [*BRY94*] ; ce mécanisme permet aux instances d'une variable d'être différenciés à divers points du programme.

L'effet de l'affectation sur l'état de la sécurité du programme est capturé par l'axiome suivante (à la Hoare [*HOA69*]) [6] :

$A_S :=$

$$\{ P[\overline{y} \leftarrow (\{x_i \mid i = 1..N\} \cup \{\overline{x_i} \mid i = 1..N\} \cup val(indirect))] \}$$

$$y := exp(x_1, \ldots\ldots, x_N)$$

$$\{ P \}$$

[5] Les constantes sont ignorées dans le calcul du flux puisqu'elles ne fournissent aucune information concernant les valeurs des variables. La variable de la sécurité d'une constante est toujours l'ensemble vide {}. Ainsi l'affectation $a := 0$ affecte $\overline{a}$ à *val(indirect).*

[6] Tous les axiomes et règles ont un ''s'' (pour sécurité) attaché à leur indice pour indiquer qu'ils sont définis sur l'état de sécurité du flux.

Où $P\ [\ a \leftarrow b\ ]$ est un prédicat[7] équivalent à P excepté que chaque occurrence libre de la variable *a* est remplacée par l'expression *b*. cet axiome généralise l'affectation avec les variables tableaux comme les commandes $a[i] := e$ et $e := a[i]$ sont équivalentes à $a := exp(a, i, e)$ & $e := exp(a, i)$ respectivement.

On note que les propriétaires de cette approche ne traitent pas le cas où y peut recevoir des informations d'une ressource qui peut-être interdite à elle, quel que soit le contenu courant en information dans cette source (i.e. un agent peut déduire des informations secrètes en effectuant une attaque interne sur le système) d'où la nécessité d'ajouter des règles et leurs sémantiques pour la non-induction des informations non autorisées qui est défini comme suit [CUP93]: un agent interne A ne peut jamais dériver des informations de haute-sensibilités à partir des informations de basse-sensibilités (à lesquelles A dispose un accès légal).

3.5.3 La composition séquentielle des commandes

La règle de la composition séquentielle noté '' ; '' est la suivante. Si la commande S1 établit un état de sécurité de flux satisfaisant un prédicat Q à partir d'un état P et S2 établit R à partir de Q, alors S1 suivi de S2 doit établir un état satisfaisant R à partir de P :

$R_{S-composition}$

$$\frac{\{\ P\ \}\ S1\ \{\ Q\ \},\ \{\ Q\ \}\ S2\ \{\ R\ \}}{\{\ P\ \}\ S1;S2\ \{\ R\ \}}$$

3.5.4 La sémantique du flux des communications

L'effet de la commande de communication (interprocessus) est d'assigner l'expression dans la commande d'envoi à la variable dans la commande de réception qui représente un flux direct. En outre, les deux ''*indirects*'' sont mis à jour pour inclure chaque autres valeurs en utilisant l'opérateur ⊔ car l'exécution des deux processus dépend maintenant du rendez-vous ayant eu lieu, c'est-à-dire, un observateur de chaque processus se rend compte que l'autre processus ait

[7] Proposition susceptible de devenir vraie ou fausse selon les valeurs attribuées aux variables qu'elle contient

communiqué (l'opérateur '!' est désigné pour l'émission, '?' pour la réception et '||' pour une exécution parallèle).

$$(P_2 \,!\, x \parallel P_1 \,?\, y, \sigma) \rightarrow (c, \sigma')$$

Où σ ressemble au σ' sauf que $\overline{y} = (\overline{x} \cup x \cup val(indirect_1) \cup val(indirect_2))$ dans σ', $indirect_1$ égale ($indirect_1 \sqcup indirect_2$) et $indirect_2$ est ($indirect_2 \sqcup indirect_1$).

3.5.5 Le parallélisme

Quand un processus fait un rendez-vous, il y a des mises à jour et des communications avec d'autres processus. Le fait que ces communications ont lieu, et que les mises à jour qui suivent sont faites, fournit des informations aux observateurs d'autres processus sur la condition qui a été rencontrée dans le processus qui a fait ce rendez-vous.

Prenez l'exemple de la figure **3.4**. Un observateur voit ce qui se produit dans le processus P2 au temps d'exécution. Puisqu'il connaît le texte de processus, il peut déduire les résultats suivants : Si la valeur de y change, alors a doit être zéro dans le processus P1 ; si $b \neq 0$ et r devient 1, P1 n'a pas pris la branche avec la commande de communication dans sa commande alternative et ainsi ($a \neq 0$) doit se tenir. Ainsi, les flux d'information indirects interprocessus peuvent se produire sans avoir lieu de communication conditionnelle.

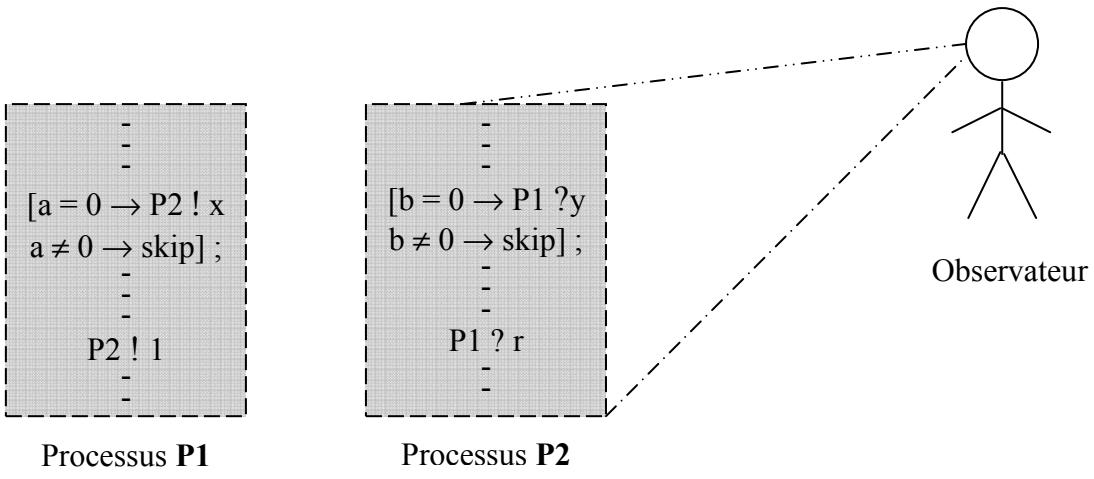


Figure 3.4 : Le parallélisme & les flux d'informations

La solution qui a été proposée est de transférer la valeur du flux d'une condition sur laquelle une communication s'exécute, seulement si la communication a lieu. Si la communication est omise, alors la valeur du flux de la condition est enregistrée dans une variable spéciale ''*rendez-vous*'' (une par processus). Sur chaque communication, *rendez-vous* est transférée dans les deux directions en tant qu'élément du flux indirect. Dans ce mécanisme, *rendez-vous* est incorporée à la variable *indirect*.

Cette approche fonctionne parce que les flux indirects interprocessus peuvent seulement signaler n'importe quelle information utile si un processus dont le flux débute, (transitivement) communique avec le processus avec lequel il devrait avoir communiqué. Avec le mécanisme de *rendez-vous*, la valeur du flux de la condition en question sera transférée quand cette communication se produit.

3.5.6 La commande alternative

Les commandes alternatives et réitérées se composent d'un ou plusieurs ''*guard*'' (i.e. branche de garde). Une ''*guard*'' est une expression booléenne, une communication ou une expression booléenne suivie d'une communication. Une *guard* est passable si, pour une expression, elle s'évalue pour vraie, et pour une communication, le processus désigné dans la commande est prêt à communiquer avec lui. Pour des *guards* se composant d'une expression booléenne et des communications, l'expression doit être vraie et le processus désigné dans la *guard* doit être prêt à communiquer pour que la *guard* soit passable.

Quand une commande alternative est exécutée, une branche dont la *guard* est passable est choisie, si plus d'une *guard* est passable, alors n'importe quelles des branches correspondantes peut être exécutée. Si aucune *guard* n'est passable alors le processus se bloque jusqu'à ce qu'une des *guards* de communication devienne passable, sauf qu'il n'y ait aucune *guard* de communication ou les processus désignés dans les *guards* sont terminés, dans ce cas la commande échoue et le processus se termine.

■■ Le cas séquentiel

Les variables dans une *guard* entrent indirectement dans la branche quand il s'exécute puisque l'exécution de la branche signifie que la *guard* doit être vraie. D'ailleurs, les *guards* de commande peuvent être connexes, ainsi une *guard* étant vraie peut impliquer la valeur d'autres *guards*. Ainsi, il y a un flux indirect des

variables de toutes les *guards* à la branche qui s'exécute. De même, si une branche n'est pas exécutée, alors ceci pourrait signifier que sa *guard* est fausse et donc d'autres *guards* sont vraies (ou faux). Ainsi, il y a un flux indirect de toutes les *guards* à toutes les branches qui ne sont pas exécutées.
Par exemple, on considère le fragment du programme suivant :

$x, y := 0, 0;$
1. $[\ \boldsymbol{B} \rightarrow x := 7;\ y := 9\ \square\ \textbf{not}\ \boldsymbol{B}\ \rightarrow\ y := 2\ ]\ ;$
2. $[\ x = 7 \rightarrow S_1\ \square\ y = 9\ \rightarrow S_2\ \square\ x = 0 \rightarrow S_3\ ]\ ;$

Figure 3.5 : Flux implicite à l'exécution de la commande alternative

Dans la deuxième commande alternative, l'exécution de n'importe quelle branche fournit des informations sur toutes les *guards*. C'est-à-dire, si S_1 s'exécute alors la condition $x = 7$ doit avoir été vraie. En conséquence, un observateur sait que y est 9 de la première instruction alternative. Cette information est également perceptible en observant que S_3 ne s'est pas exécuté. L'exécution de S_3 implique que x est zéro et que donc y est 2 puisque la deuxième branche doit être exécutée dans l'alternative précédente.

La commande alternative complète –CSP– est plus complexe que celle présentée plus tôt, pour en savoir plus, voir [*BRY95*] qui a montré sa version parallèle.

3.5.7 La commande réitérée

Sur chaque itération de la commande réitérée, la branche dont la *guard* est passable est exécutée. Si plus d'une *guard* est passable, alors comme pour l'alternative, n'importe quelle des branches est choisie. Quand aucune *guard* n'est passable, la commande se termine et le processus continue.

■■ Le cas séquentiel

Les commandes réitérées causent également des flux indirects d'information des variables dans les *guards* à toutes les variables qui pourraient probablement recevoir des flux dans le corps de la boucle. Donc un observateur de ces variables peut savoir

la valeur des *guards* en examinant les variables dans la boucle, même si la boucle ne s'exécute pas.

```
   x, r := e1(), e2();        /* expressions renvoient des
                                 valeurs non négatives */
   z, y, t := 0,0,0 ;
* [ x ≠ y → y := y + 1              /* boucle */
     □ r ≠ t → t := t + 1 ] ;
   z := 1 ;
```

Figure 3.6 : Flux implicite à l'exécution de la commande réitérée

Dans ce fragment du programme, les valeurs de y et t égaleront x et r respectivement à la fin de la boucle, même si aucune des branches ne s'exécute.
D'ailleurs, comme Reitman [*REI78*] précise, puisque toutes les variables recevant des flux directs après que la boucle fassent, ainsi à condition que la boucle se termine, les variables des *guards* de la boucle coulent indirectement dans ces variables. C'est parce qu'on connaît après l'arrêt de la boucle que toutes les *guards* sont fausses. Dans l'exemple au-dessus, en observant que z est 1, on sait que x égale y et r égale t.

Le gabarit de cette commande dans sa version parallèle est introduit dans [*BRY95*].

3.5.8 Un Système du flux d'information sain

Les axiomes et les règles donnés dans le dernier chapitre nous permettent de montrer que les systèmes écrits dans le langage de programmation CSP sont flux d'information sécurisés. Comme pour les preuves d'exactitude des programmes parallèles [*APT80*], une approche à deux-phases pour prouver que les programmes parallèles sont sécurisés est utilisée. Dans la première phase, chaque processus est prouvé individuellement avec des hypothèses faites sur les valeurs des flux d'information échangés entre les processus pendant la communication. La deuxième phase approuve ces hypothèses.

Étant donné que le système de preuve fait des hypothèses concernant les échanges de flux d'information effectués par un processus dans la première étape, dans les axiomes d'envoi et de réception, n'importe quelle post-condition peut être établie.

A_S – *envoi*

$$\{ P \} \prec \textbf{Processus} \succ ! \prec \textbf{Expression} \succ \{ Q \}$$

A_S – *réception*

$$\{ P \} \prec \textbf{Processus} \succ ? \prec \textbf{V}\text{ariable} \succ \{ Q \}$$

La convenance de la post-condition choisie est vérifiée dans la deuxième étape de la preuve en utilisant l'axiome de communication suivant :

A_S – *communication*

$$\{ z_1 = indirect_1, z_2 = indirect_2 \}$$
$$P_2 \,!\, x \parallel P_1 \,?\, y$$
$$\{ \overline{y} = \{x\} \cup \overline{x} \cup val(indirect_1) \cup val(indirect_2), indirect_1 = z_1 \sqcup z_2 , indirect_2 = z_2 \sqcup z_1 \}$$

La deuxième phase d'une preuve de sécurité du flux d'un programme CSP démontre que les preuves de chaque processus $P_1, P_2, \ldots\ldots, P_N$ coopèrent :

$R_{coopération}$

$$\frac{\parallel \{ pre_i \} \; P_i \; \{ post_i \} \; i = 1..N \;\; \text{co-operate}}{\{ \wedge pre_i \mid i = 1..N \} \; P_1 \parallel P_2 \parallel \ldots\ldots \parallel P_N \; \{ \wedge post_i \mid i = 1..N \}}$$

Les processus coopèrent si les deux conditions suivantes se tiennent :

1. Les prédicats utilisés dans la preuve du processus P_i ne contiennent aucune variable libre modifiable dans n'importe quel autre processus P_j.
2. Pour toutes les communications qui sont couplées sémantiquement c.-à-d.
   $\{ p_i \} \; P_j \,!\, \prec \textbf{Expression} \succ \{ q_i \}$ et $\{ p_j \} \; P_i \,?\, \prec \textbf{V}\text{ariable} \succ \{ q_j \}$

on a :

$$\{ p_i \wedge p_j \} \; P_j \,!\, \prec \textbf{Expression} \succ \parallel P_i \,?\, \prec \textbf{V}\text{ariable} \succ \{ q_i \wedge q_j \}$$

Notez comment la 2ème condition indique que les communications *sémantiquement* réunies, c'est-à-dire, les commandes qui appellent chaque autre processus et qui peut probablement communiquer pendant l'exécution du programme. Il peut sembler étrange que des commandes *sémantiquement* accouplées soient énoncées dans la règle puisque l'état de sécurité de flux ne capture pas les commandes sémantiquement assorties. C'est un autre secteur où une analyse statique de sécurité est facilitée par une analyse fonctionnelle.

Exemple

Comme exemple de la manière dont le système de preuve travail, on considère un programme où le processus ***A*** envoie des informations secrètes au processus ***C*** par l'intermédiaire du processus ***B***. Le code et les prédicats que nous voulons prouver sont donnés sur la figure **3.7**.

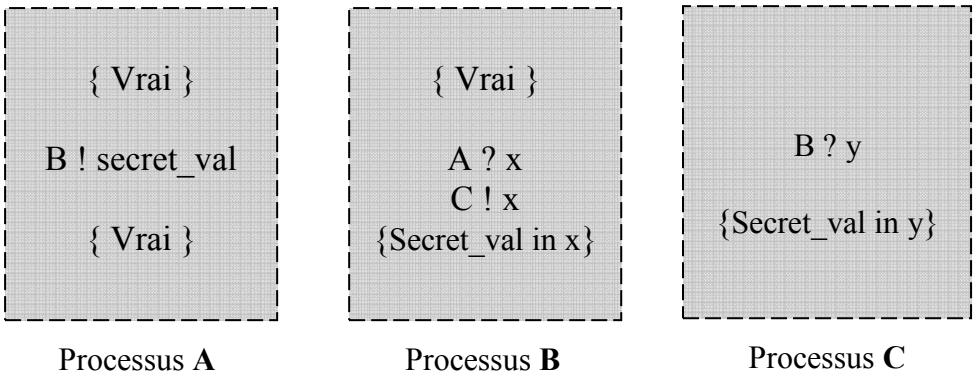


Figure 3.7 : Exemple simple de preuve de flux d'information

Nous devons pouvoir prouver que le processus ***C*** reçoit des informations interdites (i.e. que le prédicat $secret\_val \in \overline{y}$ est vrai). Puisque toutes les commandes de chaque processus sont des communications simples, chaque processus est individuellement prouvé en utilisant le fait que les axiomes des commandes de communication permettent n'importe quelle post-condition, et ainsi la première phase de la preuve se termine. Notez également que puisqu'aucun processus ne contient une déclaration conditionnel, *indirects* de chacun sont vides.
L'axiome de communication signifie que le prédicat $\overline{x} = \{secret\_val\}$ doit se tenir après la première commande dans le processus ***B*** ; cet axiome impose également que le prédicat $\overline{y} = \{secret\_val, x\}$ se tienne après la commande de communication dans le processus ***C***. Comme le dernier prédicat implique la post-condition choisie, le programme soit avéré.

3.6 Conclusion

Le contrôle de flux d'information est largement adopté, amélioré et déployé dans différents domaines. C'est un mécanisme nécessaire pour assurer la confidentialité et l'intégrité de bout en bout dans les systèmes d'infrastructure critique tels que les réseaux scada où très peut de travaux s'inscrivent dans ce contexte.
Dans ce chapitre, nous avons introduit les concepts du contrôle de flux d'information, nous avons cité quelques travaux concernant ce mécanisme de sécurité et s'inscrivant dans différents domaines puis nous avons présenté le modèle de contrôle de flux d'information adapté pour notre modèle détaillé dans le dernier chapitre et la nécessité de renforcer ce modèle contre l'induction interne des informations secrètes qui peuvent être utilisées pour endommager le fonctionnement du système.

CHAPITRE

4

Une Approche Intégrée pour La Sécurité des Systèmes SCADA (AI3S)

4.1 Introduction

De nos jours des attaquants sont engagés dans le développement de nouvelles techniques pour injecter ou activer les défauts secrets dans les systèmes existants [BAN07], qui implique que les menaces évoluent pendant la vie des systèmes exposés aux attaques malveillantes. Ceci signifie que de tels systèmes doivent également évoluer pour que les attaques ne lancent jamais des échecs de système. Le but idéal d'une telle évolution serait l'élimination complète des vulnérabilités (éliminant de ce fait toutes chances d'une attaque causant un échec), mais il est bien connu qu'un tel but est très difficile, sinon impossible, à réaliser. Néanmoins, et dans ce rapport, nous allons proposer une approche intégrée pour la sécurité des informations dans les ICs[1]. Cette approche consiste en un modèle de tolérance d'intrusion (Intrusion-tolerant) et de contrôle de flux d'information (CFI) amélioré.

[1] **I**nfrastructures **C**ritiques (i.e. Les Systèmes SCADA, PCS, etc.)

Nous allons définir les propriétés et les exigences particulières pour l'TI[2] et le contrôle de flux d'information, nous allons montrer à travers des scénarios leur nécessité et nous allons présenter notre approche intégrée pour la sécurité des systèmes SCADA.

4.2 TI et CFI pour les systèmes d'infrastructures critiques

Un système résilient doit traiter tels défauts/intrusions, qui peuvent être masqués par des protocoles insensibles aux défaillances byzantins BFT[3] (par exemple, [*CAS02*], [*MON06*], [*KOT07*]). Ces protocoles sont typiquement courus sur les systèmes repliés et peuvent tolérer l'échec d'un ensemble fini de f répliques. De tels protocoles ont une utilité limitée dans les systèmes ayant une longue durée de vie où les ennemis malveillants déploient constamment des attaques et causent des intrusions. En fait, les protocoles tolérants aux intrusions sont utiles pour retarder la corruption du système replié global, par un certain intervalle du temps qui dépend de la valeur réelle de f et sur la façon de création de différentes répliques [*OBE06*]. Malheureusement, tandis qu'une valeur plus élevé de f laisse tolérer plus de répliques compromises, cela signifie également qu'il y a plus des répliques qui doivent être différentes entre eux.

Le contrôle de flux d'information peut être considéré comme un complément de tolérance d'intrusion, le CFI assure la confidentialité des informations même après leur dissémination. Ainsi, il est fortement lié aux mécanismes de sécurité du niveau transport pour l'échange des informations.

Le CFI et le TI pour les systèmes SCADA doivent vérifier certaines conditions :

- Assurer la vivacité et la disponibilité de l'IC;
- Fournir un support des politiques diversifiés des organisations;
- Informations contrôlées avec différents niveaux de granularité;
- Fournir des canaux de communication sécurisés;
- Gestion des relations de confiance entre les différents modules de CFI et de TI;
- Une approche décentralisée pour leur gestion et application.

[2] **T**olérance d'**I**ntrusion
[3] **B**yzantine **F**ault-**T**olerant

4.3 Scénarios

Afin de montrer la nécessité de sécuriser les informations d'un système d'infrastructure critique nous avons considéré les scénarios suivants :

■■ Scénarios 1 : (Attaque externe)

Nous avons vu au chapitre **2** (cf. § **2.5.2.1**) que chaque réplique CIS est déployée dans un système d'exploitation différent (par exemple, Linux, FreeBSD, Windows XP), et les systèmes d'exploitation sont configurés pour employer différents mots de passe et différents pare-feux internes (par exemple, iptables, ipf, pare-feu de Windows). Le *wormhole* est synchrone et assez sécurisé qu'il peut déclencher à l'heure le recouvrement proactif et exécuter un protocole simple de vote qui produit un MAC pour un message si au moins f+1 répliques l'approuvaient (si le *wormhole* est sécurisé, aucune réplique malveillante peut le forcer à signer un message non approuvé).

Nous avons vu aussi que l'en-tête d'authentification (AH) du protocole IPSEC [*KEN05*] fonctionne dans les ordinateurs stations et dans les répliques CIS. L'idée fondamentale est que les ordinateurs stations accepteront seulement des messages (expédiés à ses destinations par une réplique distinguée : le leader), avec un code d'authentification (MAC) valide, qui peut seulement être produit si le message est approuvé par f +1 répliques différentes.

En effet ce modèle ne traite pas le souci où un attaquant peut causer une fuite d'informations voire une catastrophe grâce à une réplique malveillante. Ainsi nous considérons les cas suivant :

Approbation malveillante : Une réplique défectueuse approuve un message illégal ; i.e. Elle vote qu'un faux message est illégal, la réplique défectueuse est imposée, ce qui fait une intrusion, parce que toutes les répliques correctes vérifient la même politique de sécurité.
Suspect malveillant : Une réplique défectueuse signale au *wormhole* une accusation concernant une réplique correcte ; la réplique défectueuse est imposée, parce qu'une réplique correcte ne montre aucun comportement incorrect.

Dans ces deux cas, la réplique défectueuse est imposée, elle est toujours considérée correcte et par conséquence deux situations peuvent être distingués :

– La réplique défectueuse est simple : dans ce cas l'attaquant peut utiliser les capacités de cette réplique (i.e. Encryptions-décryptions, mots de passe …etc.) qui mène à une fuite d'information ce qui met la globalité de système en danger (le hacker sauvegarde une copie de message avant de l'envoyer au service de signature, l'attaquant sur ce chemin peut briser l'infrastructure en divulguant ses données).

– La réplique défectueuse est leader : le problème le plus important, le leader peut faillir avec malveillance, par conséquence il expédie des messages illégaux au LAN (les ordinateurs stations typiquement ne peuvent pas distinguer les répliques défectueux des répliques correctes), ou de manière bénigne, se brisant, restant silencieux (inonde le réseau avec des paquets arbitraires) ou même devenu trop lent, ainsi l'attaquant sur ce chemin envahit l'IC et peut produire des situations où leur impact socio-économique peut être énorme.

■■ Scénarios 2 : (Attaque interne)

Nous avons noté au chapitre **3** (cf. § **3.5.2**) que le modèle de CFI adopté pour notre approche ne traitent pas le cas où une variable (i.e. toute sorte de stockage d'informations) peut recevoir des informations d'une ressource qui peut-être interdite à elle, quel que soit le contenu courant en information dans cette source (i.e. un agent peut déduire des informations secrètes en effectuant une attaque interne sur le système) d'où la nécessité d'ajouter des règles et leurs sémantiques pour la non-induction des informations non autorisées pour lesquelles un agent interne A peut dériver des informations de haute-sensibilités à partir des informations de basse-sensibilités (à lesquelles A dispose un accès légal).

Pour plus de rigueur on prend le modèle de sécurité multi-niveaux [*CUP93*], soit A un sujet (agent), Clear(A) son autorisation et soit i être une portion d'information, Clas(i) sa classification. Si Clas(i) ≤ Clear(A) alors $\{i\} \rightsquigarrow A$, i.e. A est explicitement autorisé pour savoir l'information i. Dans l'autre coté, si $\neg$ (Clas(i) ≤ Clear(A)) alors $\{i\} \not\rightsquigarrow A$, i.e. A est explicitement interdit de connaître l'information i.
Nous considérons un exemple [*CUP93*] illustrant les vulnérabilités introduites par le problème d'induction des informations secrètes.

Considérons une base de données à multi-niveaux employée par les agents A (non classifié) et B (secret). Supposons que B crée une session secrète et insère le tuple

secret Dest(Enterprise, Rigel). En ce formalisme, nous pouvons exprimer ce fait en précisant qu'il y a une entrée secrète ***IN***, tel que :

Val (***IN***, "**Insert** Dest(Enterprise, Rigel)")

Puis, ***B*** crée une session non classifiée et insère dans la base de données le tuple Dest(Enterprise, Restricted). En ce formalisme, nous pouvons exprimer ce fait en précisant qu'il y a une entrée non classifiée ***IN'***, tel que :

Val (***IN'***, "**Insert** Dest(Enterprise, Restricted)")

La valeur restreinte "Restricted" est une valeur particulière employée par Jajodia et Sandhu dans [SAN92] pour dire à des agents que l'information existe mais elle est secrète. En conséquence, quand l'agent ***A*** pose la question « quelle est la destination de l'entreprise », il obtiendra la réponse limitée "Restricted". Par conséquent, ***A*** peut déduire que ***B*** a exécuté une entrée secrète afin d'insérer la destination de l'entreprise. Ainsi, ***A*** sait qu'une entrée secrète ***IN*** existe, tels que :

¬Val (***IN***, Null): est valide.

Supposons que nous ajoutons la relation suivante :

SR(Starship , Porté-d'action)

Considérons maintenant l'exemple suivant de la relation SP [*CUP93*] :

Star-ship	Porté-d'action	Classification
Enterprise	20000	Non-classifié

Cet exemple indique que « la porté d'action de l'entreprise est 20000 » est une information non secrètes parce qu'elle est liée aux caractéristiques techniques de l'entreprise et peut-être qu'elle est largement distribuée et bien connue. Maintenant, supposons que ***B*** insère le tuple secret Dest(Enterprise, Rigel). En connaissant la porté d'action, ***A*** peut éliminer les destinations trop éloignées (voir la figure **4.1**) en réalisant des dérivations plausibles. Par exemple, si Talos est une destination plus de 20000 distant, alors ***A*** sait (voir [*GAR92*] pour un exemple plus complet) que :

¬ Dest(Enterprise, Talos) : en éliminant une possibilité.

La figure **4.1** illustre comment un agent non classifié peut exploiter des vulnérabilités découvertes en poursuivant le raisonnement du dernier paragraphe.

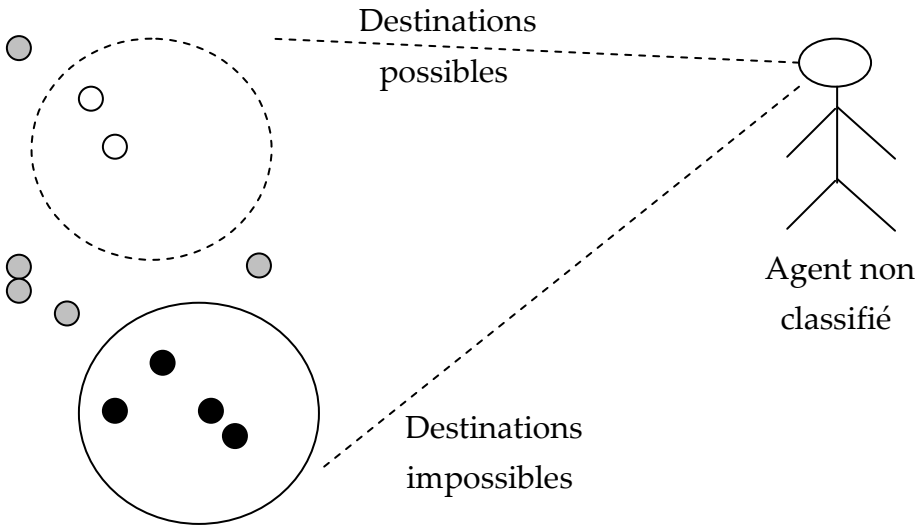


Figure 4.1 : Induction basée sur l'élimination de possibilités.

Ces scénarios, renforcés par plusieurs incidents récents (cf. § **1.6**), produit d'un grand souci concernant la sécurité de ces infrastructures, particulièrement au niveau de gouvernement.

La section suivante va présenter une solution de ces problèmes basée sur le contrôle de flux d'information imposé à chaque réplique de CIS et à chaque sous réseau des ordinateurs stations (LAN).

4.4 Présentation de la solution

Le travail qui a été proposé par Bryce, Banâtre & Le Métayer [*BRY95*] pour l'analyse de flux d'information interprocessus, consiste en une analyse du code des programmes, ils ont définie une sémantique du flux pour un langage parallèle et développé un système de preuve axiomatique à partir de cette sémantique. Ce système de preuve permet à la sécurité des systèmes parallèles d'être vérifiée en validant des attributs sur les flux d'information entre les variables qui sont insérées dans le code du programme. Après notre étude détaillée et vérification de la sûreté

du modèle proposé dans [*BRY95*] nous avons noté des lacunes au niveau des définitions des axiomes concernant la non-induction d'informations.

Dans ce travail, nous présentons une nouvelle approche intégrant un modèle de contrôle de flux d'information et un modèle de tolérance d'intrusion pour la protection des infrastructures critiques qui permet la gestion de la confiance et la disponibilité de l'IC.

L'amélioration de modèle du contrôle de flux d'information concerne essentiellement la définition des axiomes et leur sémantique pour la non-induction des informations (en empêchant toute attaque interne) afin de réaliser des flux d'information sains pour assurer la non-divulgation d'informations.

Le modèle de tolérance d'intrusion CIS intégré (comme un pare-feu) permet à la fois la protection de l'IC et l'affranchissement du LAN du trafic superflu grâce à sa méthode d'acheminement des messages vers ses destinations (i.e. consolidation par la source).

Le choix du CIS est justifié par : (ces caractéristiques ont été discutées dans le chapitre **2**)

- La fourniture d'un pare-feu distribué (Distributed firewall);
- Un pare-feu au niveau application (Application-level firewall) ;
- Un modèle riche en contrôle d'accès (Rich access control model) ;
- Un pare-feu tolérant aux intrusions (Intrusion-tolerant firewall).

Le modèle de sécurité proposé permet à un système d'IC (i.e. grille d'énergie électrique, pétrole, eau, gaz, etc.), de maintenir sa vivacité et disponibilité et de contrôler les flux d'informations à travers l'IC en empêchant des attaquants (internes ou externes) de divulguer les informations sensibles de système.

La solution consiste en un ensemble de définitions, de règles, de prouve de sûreté, pour la protection des informations et la globalité du système et des algorithmes permettant leur implémentation.

4.5 L'approche hybride pour la sécurité des systèmes SCADA

Le principe du modèle de tolérance d'intrusion est –en utilisant les mécanismes de rétablissement– d'augmenter la résilience de n'importe quel système replié (intrusion-tolérant), ce système est capable de tolérer jusqu'à f intrusions : un nombre illimité d'intrusions peut se produire pendant sa vie, tant que pas plus que f

ne se produisent entre les rajeunissements. Notre modèle illustré dans la figure **4.2** garde les mêmes concepts (cf. § **2.5.2**).

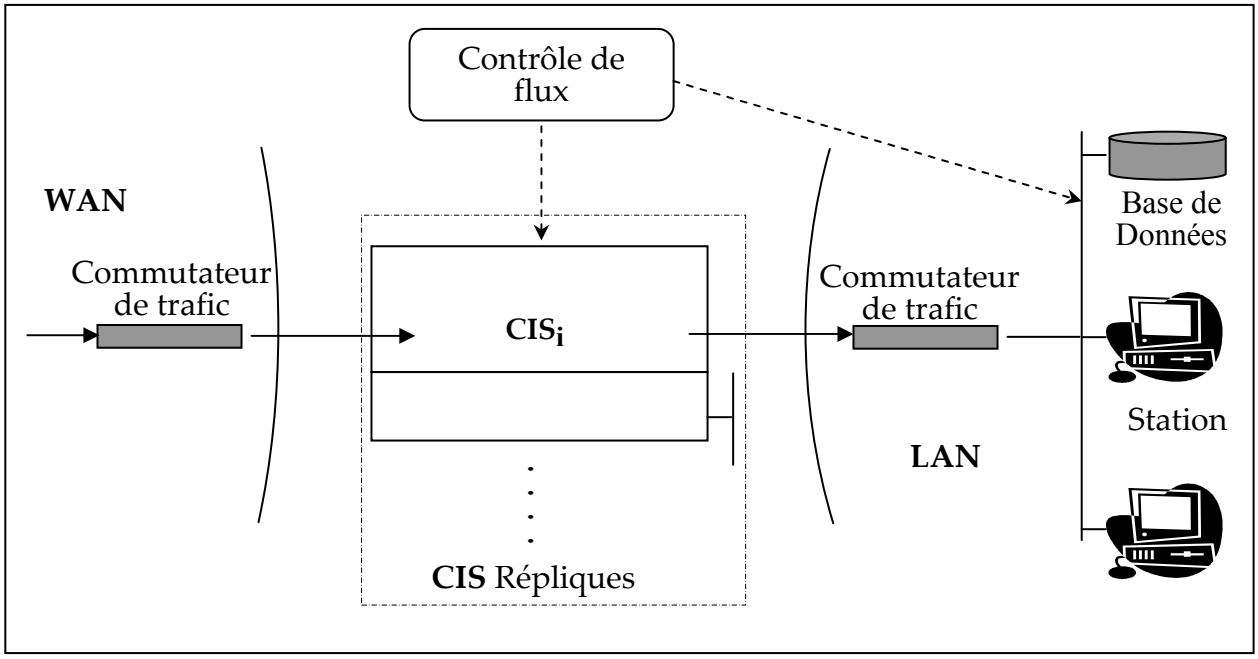


Figure 4.2 : Le modèle de TI et de CFI

Les hypothèses et les algorithmes du modèle proposé seront énoncés dans le chapitre suivant.

Dans notre approche hybride, nous nous basons sur le modèle précédant pour garantir la disponibilité et la vivacité du système (IC) et le modèle de CFI pour empêcher la divulgation des informations sensibles, en interdisant un attaquant dissimulé (hacker ou agent) de dévoiler les données de l'IC. Ainsi, notre modèle étend le modèle de TI en utilisant les règles de contrôle de flux d'information pour permettre la confidentialité et l'intégrité des informations de l'infrastructure même après leur dissémination.

4.5.1 Le modèle de tolérance d'intrusion TI

Nous envisageons la TI comme service qui peut être employé pour augmenter la résilience des systèmes repliés (intrusion-tolérants) en permettant à ces systèmes de tolérer un nombre arbitraire de défauts. Un but si ambitieux est réalisé par la combinaison de deux importantes caractéristiques complémentaires :

■■ Rétablissement

Le modèle TI permet à un système intrusion-tolérant de récupérer des actions/défauts malveillants passés, en nettoyant les effets de telles actions par (i.) des rétablissements périodiques temps-déclenchés, et (ii.) des rétablissements réactifs événement-déclenchés quand un comportement malveillant est détecté. Quand plus de f répliques détectent qu'une réplique donnée R_i est sans équivoque défectueuse (par exemple, R_i envoie des messages signés contradictoires dans un protocole distribué), cette réplique est immédiatement récupérée (c'est un rétablissement événement-déclenché).

Si, d'une part, les répliques seulement suspectent que R_i soit défectueuse (mais elles ne peuvent pas être sûres), le rétablissement de R_i est programmé de telle manière que la disponibilité de système global ne soit pas affectée. En outre, des rétablissements temps-déclenchés (proactifs) sont exécutés périodiquement afin de neutraliser les effets de tous les défauts et intrusions non détectés.

■■ Évolution

Cette fonctionnalité du modèle TI déclenche le rétablissement d'une certaine réplique, elle modifie les vulnérabilités qui peuvent être exploitées par un ennemi malveillant (par exemple, les mots de passe d'accès d'OS, des ports communicatifs, des méthodes d'authentification, etc.).

De plus, cette fonctionnalité permet des mises à niveau en ligne (par exemple, installation des pièces de sécurité, installation d'une nouvelle version de l'application intrusion-tolérante étant protégée) avec un impact minimum sur la disponibilité [*OBE06*].

Ces caractéristiques rendent ce modèle différent de n'importe quel autre service de rétablissement proposé dans le passé [*CAS02*], [*ZHO02*].

4.5.2 Le modèle de contrôle de flux d'information

Une approche basée sur les travaux de Denning [*DEN76*] pour l'analyse de flux d'information des programmes a été proposé par Bryce, Banâtre & Le Métayer [*BRY95*].

Le problème de base adressé par cette approche est que la plupart des systèmes emploient les logiciels du domaine public et que les violations de sécurité peuvent se produire non seulement par la lecture illégale de l'information, mais aussi par l'inférence des informations confidentielles à partir d'autres informations en utilisant

la connaissance de code du programme, l'approche adoptée pour aborder ce problème est basée sur l'analyse du code, où la sémantique du flux d'information pour un langage de programmation parallèle est définie, et un système de preuve axiomatique est développé à partir de cette sémantique. Ce système de preuve permet à la sécurité des systèmes parallèles d'être vérifiée en validant des attributs sur les flux d'information entre les variables qui sont insérées dans le code du programme.

Après vérification du modèle proposé dans [*BRY95*] nous avons noté des lacunes aux niveaux des définitions des sémantiques des flux concernant la non-induction d'information à l'aide des exemples pour les démonstrations de sûreté. Pour cela, nous avons proposé une amélioration pour ce modèle en présentant de nouvelles règles, stratégie (avec un exemple détaillées en montrant son efficacité) et de nouvelles définitions et axiomes. Les règles de contrôle de flux que nous avons proposées sont utilisées pour réduire le champ de vulnérabilité touchant les systèmes d'infrastructure critique (i.e. SCADA, PCS[4], etc).

4.6 L'architecture distribuée de l'approche AI3S

L'architecture distribuée du service de protection CIS exige une architecture distribuée du système de contrôle de flux d'information.
Le système de tolérance d'intrusion et de contrôle de flux d'information est un ensemble de modules se coopèrent pour protéger l'IC et garantir sa disponibilité. Nous supposons que ces modules partagent une relation de confiance (i.e. certifiée par une autre autorité).
Le contrôle est effectué au moment :

- De la réception d'un message (requête) par chaque réplique du CIS.
- Avant et après le flux d'information entre les objets de système (i.e. répliques), par exemple : au moment d'assignation de variables (i.e. composantes du message), pendant l'exécution des opérations de communication inter-objets.
- De lancement de chaque requête interne vers le système de fichier dans les sous-stations (LANs) de l'IC.

La figure **4.3** montre l'architecture logique distribuée du modèle AI3S.

[4] PCS : Process Control System

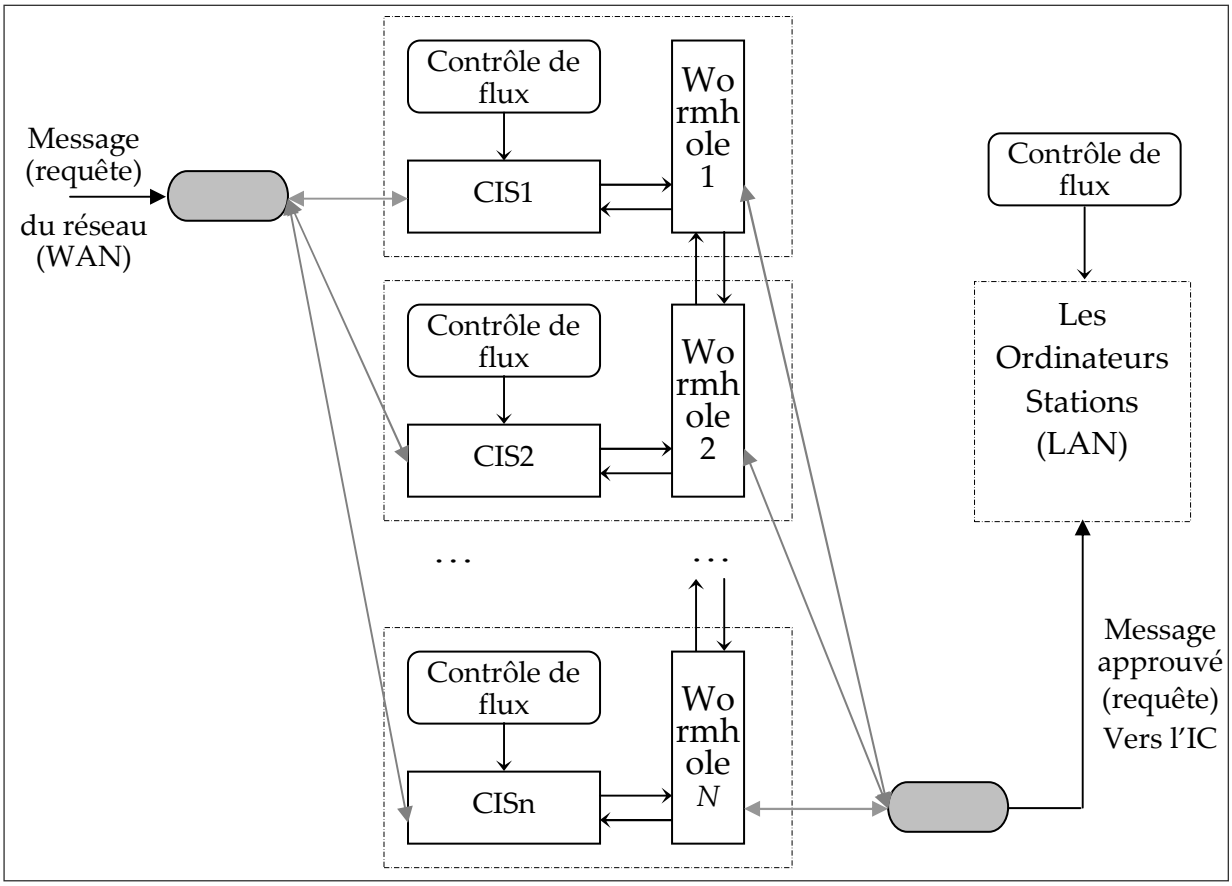


Figure 4.3 : L'architecture distribuée d'AI3S

Chaque requête d'accès (i.e. message envoyé) à une destination de l'IC doit passer sous la surveillance du module de contrôle de flux d'information soit en sa réplique ou sa sous-station (LAN) correspondante.

4.7 L'architecture de la sécurité locale avec AI3S

L'architecture locale d'AI3S se décompose en deux modules l'un consacré à la protection de l'IC contre les menaces extérieures (implémenté sur chaque réplique) qui est composé de trois fonctions fondamentales : la fonction d'authentification d'accès, la fonction de contrôle de flux et la fonction de vote et de recouvrement proactif-réactif. La figure **4.4.*a*** montre l'architecture de la sécurité locale à chaque réplique du Cis-firewall.

i) L'authentification d'accès

1.*a*. L'agent distant lance une requête (message) orientée vers une destination de l'IC. Le système de contrôle d'accès (OrBAC) effectue une authentification.

2.*a*. Si l'agent est refusé, la requête est échouée et par conséquence le système de contrôle d'accès signale un message d'erreur à l'agent.

3.*a*. Si l'agent est accepté, le message est affranchi vers CIS réplique.

ii) *Le contrôle de flux*

4.*a*. A la réception de message par la couche application de CIS un contrôle de flux est imposé sur la manipulation de ce message (i.e. interdire la copie ou la lecture du contenue de message) afin d'empêcher la divulgation d'informations si la réplique est dissimulée.

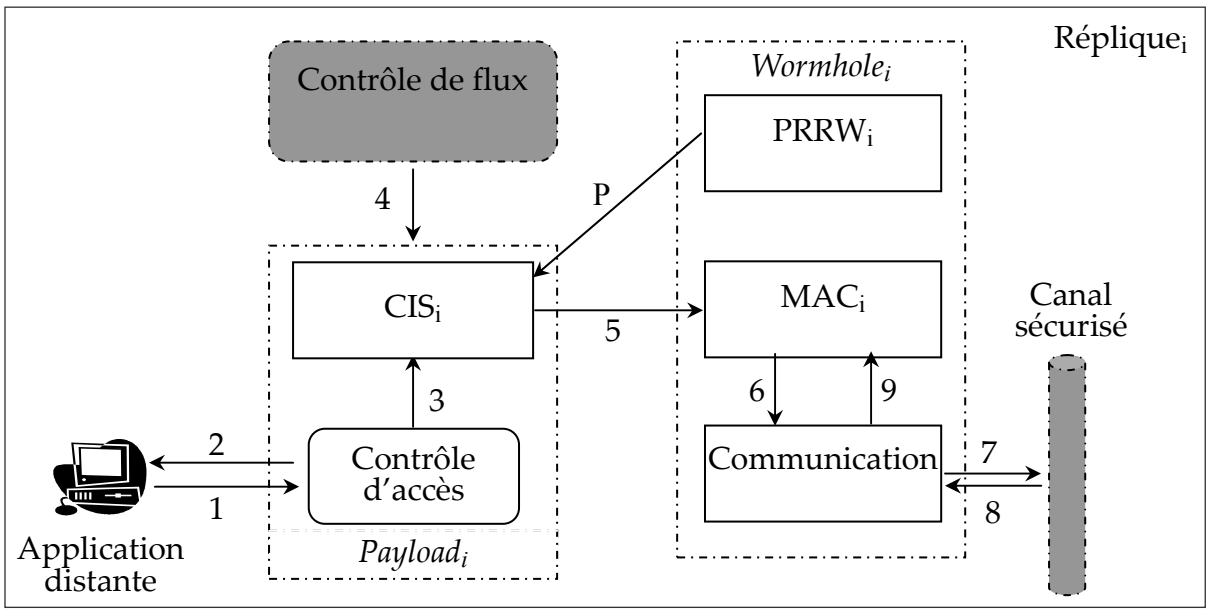
5.*a*. Le CIS-réplique juge le message et envoi le résultat au protocole de vote. A chaque action de flux d'information, les variables correspondantes sont mises à jour.

ii) *Le vote et le recouvrement proactif-réactif*

6 & 7.*a*. Le protocole de vote effectue un broadcast de jugement de sa CIS-Payload vers les n répliques de CIS.

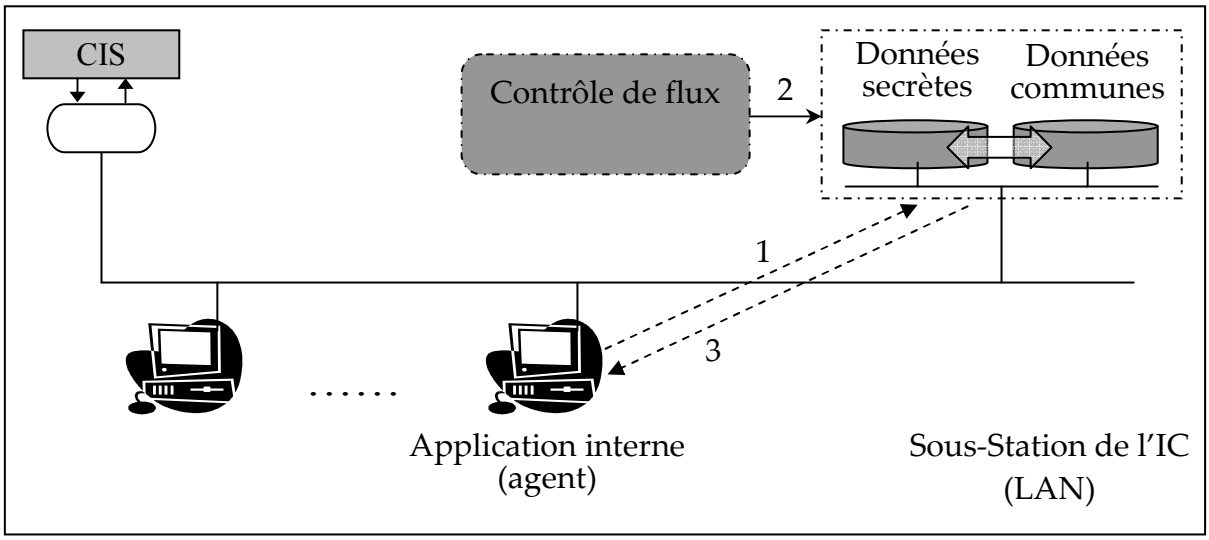
8 & 9.*a*. Et récolte les jugements des autres répliques concernant le même message, si au moins f+1 CIS-Payload l'approuvaient, le protocole IPSEC/AH[5] génère un MAC (*Message Authentication Code*) pour ce message puis va l'envoyer à sa destination s'il s'agit d'un leader.

Finalement la transition P s'agit des actions de rétablissements périodiques (restauration proactive), et rétablissements réactifs quand un comportement malveillant est détecté.



(a) : Renforcement de la sécurité externe

[5] **I**nternet **P**rotocol **Sec**urity /**A**uthentication **H**eader



(b) : Renforcement de la sécurité interne

Figure 4.4 (*a*&*b*): Architecture de sécurité

L'autre module est consacré à la protection de l'IC contre les menaces internes (implémenté sur chaque sous-station : LAN) qui consiste en une fonction principale, le contrôle de flux d'information en empêchant l'induction des informations non autorisées :

1.*b*. Un agent interne lance une requête vers le système de fichiers (Base de Données).
2.*b*. A la réception de la requête le SGBD répond selon un mécanisme de contrôle de flux.
3.*b*. Si l'information recherchée est commune (de basse-sensibilité) le SGBD la libère à l'agent propriétaire de la requête, si elle est secrète (de haute-sensibilité) le contrôle de flux vérifie Si Clas(***info***) ≤ Clear(***agent***) alors $\{\boldsymbol{info}\} \rightsquigarrow \boldsymbol{agent}$, i.e. le SGBD libère l'information recherchée à l'agent. Sinon le SGBD répond par échec à sa requête.

La figure **4.4.*b*** montre l'architecture de la sécurité locale à chaque sous-station de l'IC.

4.8 Conclusion

Nous avons donné dans ce chapitre une présentation globale de notre approche hybride pour la protection des infrastructures critiques (ICs). Nous avons expliqué les spécificités du modèle TI (en anglais : *Intrusion-Tolerant model*) et du contrôle de flux d'information du modèle AI3S et nous avons montré comment ces deux mécanismes de sécurité sont liés pour obtenir un modèle de protection de bout en bout fiable pour les systèmes SCADA, PCS, etc. nous avons également fourni une architecture logique pour le système de sécurité.
Le développement et la modélisation du modèle AI3S que nous avons proposé feront l'objet du prochain chapitre.

CHAPITRE

5

Développement De l'Approche AI3S

5.1 Introduction

Le modèle intégré pour la sécurité des infrastructures d'informations critiques (AI3S) proposé dans le quatrième chapitre consiste en une intégration de deux mécanismes de sécurité à savoir la tolérance d'intrusion et le contrôle de flux d'information.

Dans ce chapitre, nous développons le nouveau modèle en décrivant formellement les composantes de l'AI3S, ses politiques de protection, ses règles et preuves de contrôle de flux d'information et des algorithmes (pseudo-codes) permettant leur implémentation.

5.2 Le modèle AI3S

L'approche AI3S garde les même concepts de base des deux modèles ''tolérance d'intrusion & contrôle de flux d'information'', elle étend le deuxième modèle en ajoutant des nouvelles règles et leur sémantique (fournissant une protection interne d'un système SCADA), ensuite l'applique au premier modèle en réalisant leurs intégration (fournissant une protection externe). Les sections suivantes présentent un achèvement de ces soucis.

5.2.1 Le module de la protection externe

Dans ce qui suit, nous énonçons les pseudo-programmes permettant l'implémentation des fonctions fondamentales consacrées à la protection de l'Infrastructure Critique contre les menaces extérieures, puis nous présentons un système de communication sécurisé pour les systèmes SCADA (en résolvant les problèmes issus du premier scenario exhibé au chapitre précédant), en effet, on peut modéliser le système Cis-firewall (cf. § **2.5.2.2**) par un ensemble de processus (chaque réplique sera modélisée par plusieurs processus séquentiels qui s'exécutent en parallèle ''||'' en utilisant un langage dite CSP), qu'on peut vérifier la sûreté des flux interprocessus en se basant sur des systèmes de preuve de sécurité développés dans [*BRY95*] et [*KRZ79*] (On ne s'intéresse pas ici de cette vérification, elle pourra faire l'objet d'un travail future) .

CSP (**C**ommunicating **S**equential **P**rocesses) est un langage de programmation parallèle développé par Tony Hoare, ce langage est analysé pour le flux d'information dans [*BRY95*]. Les déclarations principales du langage sont données dans la figure **5.1**.

Prog	::=	[label::Process ||…..|| label::Process]	programme
Process	::=	Decl≺ ; Decl ≻; Cmd ≺ ; Cmd ≻	processus
Decl	::=	**var** v | **array** v	déclarations
Cmd	::=	Comms | Alt | Rep | skip | v:= E | Cmd; Cmd	commandes
Comms	::=	send | receive	communication
send	::=	label ! E	émission
receive	::=	label ? V	réception
Alt	::=	[guard → Cmd≺ ; □ guard → Cmd ≻]	alternative
Rep	::=	*[guard → Cmd≺ ; □ guard → Cmd ≻]	répétitive
guard	::=	B | Comms	guard

Figure 5.1: La syntaxe des déclarations CSP

Sur la figure, ≺ ≻ signifie zéro ou plusieurs répétitions des unités syntaxiques incluses, "V" représente une variable ou une liste de variables, "E" pour une expression entière et "B" pour une expression booléenne.

On se réfère à cette syntaxe pour décrire les algorithmes proposés contrôlant la communication entre une application distante et les stations de l'infrastructure

critique, en fournissant des preuves formelles de sûreté et en montrant comment un contrôle de flux puisse résoudre les menaces des **Approbations malveillantes** et **Suspects malveillants** (scénario 1)**,** décrites dans le chapitre **04**.

La figure **5.2** localise sur chaque réplique les trois algorithmes fondamentaux (ATM, ARPR, ASI) qui assurent le service Cis-firewall.

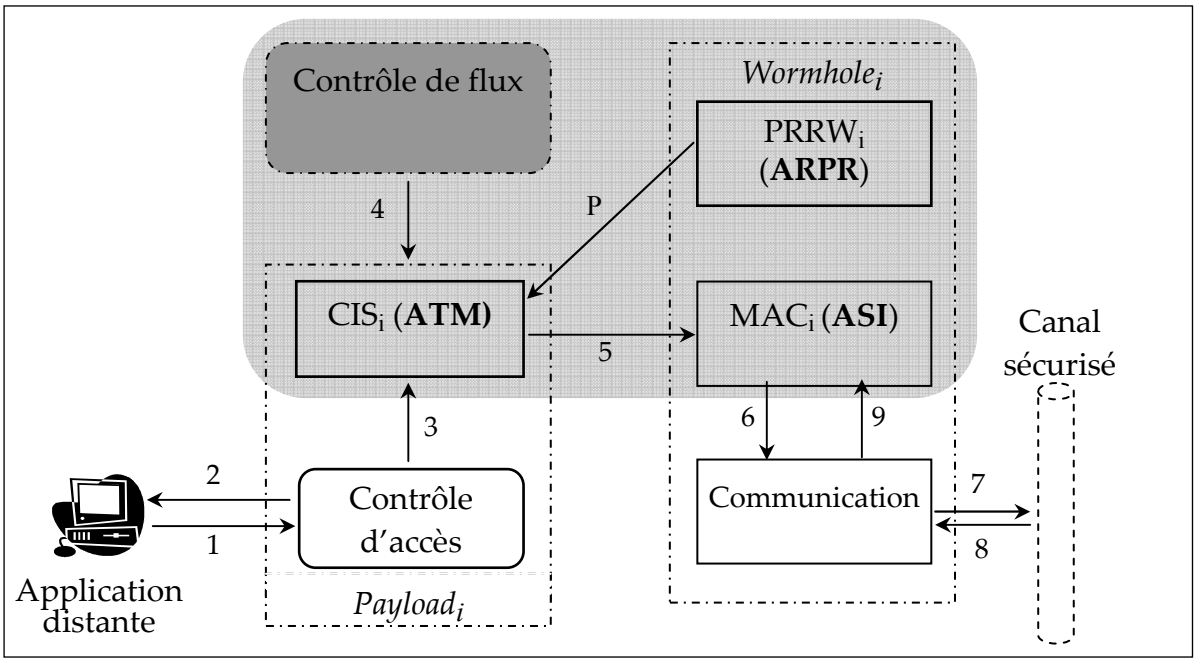


Figure 5.2 : Les trois algorithmes *ATM*, *ARPR* et *ASI* sur chaque réplique

5.2.1.1 Implémentation du service Cis-firewall en CSP

Cette section présente une implémentation en CSP (Communicating Sequential Process) du service Cis-firewall et des démonstrations en utilisant des théorèmes que chaque algorithme proposé (qui participe au fonctionnement de ce service) est sain (figure **5.2**), ces algorithmes sont exécutés en parallèle (dont chaque commande est subie d'un contrôle de flux selon le modèle de contrôle de flux) comme l'indique le pseudo-code suivant :

Cis-firewall[i] :: [ATM || ASI || ARPR]

I) L'algorithme de traitement de message (ATM)

C'est l'algorithme principal, il est exécuté par le CIS pour traiter les messages entrants du WAN au LAN protégé. Le même algorithme est employé pour

manipuler les messages venant de la direction opposée (probablement avec une politique plus détendue).

La vérification de politique (contrôle d'accès) est faite dans un moteur de politique accédé par la fonction *Pol_verify*. Les interactions entre une réplique CIS_i et son Wormhole local W_i sont faites par une interface bien définie, offrant les services suivants :

- *W_sign(m) :* retourne un MAC pour le message *m* obtenu avec la clef partagée *K*, si et seulement si au moins f +1 répliques appellent *W_sign(m)* en leurs Wormholes locaux ;
- *W_verify(m_σ)* : retourne vrai si σ est un MAC de *m* produit en utilisant *K*, et faux autrement ;
- *W_leader ()* : retourne l'identificateur de la réplique leader courante;
- *W_suspect_silent(j)* et *W_suspect_malicious(j)* : informe le Wormhole, respectivement, que la réplique CISj semble être silencieuse ou qu'elle a exécuté une certaine action malveillante.

■■ Le Pseudo-code de l'algorithme ATM en CSP :

ATM :: [P-receive(WAN, m) || P-receive(LAN, mσ) || Periodic]

P-receive(WAN, m):: % admettre un contrôle de flux de la
1: [Pol_verify(m) → % commande alternative.
2: Vote := Vote ∪{m};
3: σ := W_sign(m); % voter un message
4: Vote := Vote \{m};
5: [m_σ ∈ Transmis →
6: Transmis := Transmis \ {m_σ} % un message qui est déjà transmis
7: □ m_σ ∉ Transmis →
8: En_attente := En_attente ∪ {m_σ };
9: □ [leader = i → P-multicast(LAN, m_σ)]]] % transmission de messages

P-receive(LAN, m_σ):: % admettre un contrôle de flux de la
10: [m_σ ∈ En_attente → % commande alternative.
11: En_attente := En_attente \ {m_σ };
12: □ [W_verify(m_σ) → % le message est valide
13: Transmis := Transmis ∪ {m_σ };
14: □ ¬ W_verify(m_σ) → % le message n'est pas valide et par conséquence
15: W_suspect_malicious(sender(m_σ))]] % l'expéditeur est malveillant.

```
Periodic :: *[ Tv ; Vote ≠ ∅ →                    % admettre un contrôle de flux de la
16:     P-multicast(WAN, Vote);                    %  commande repetitive.
17:       □ Tf; mσ ∈ En_attente →
18:           P-multicast(WAN, m);
19:           leader_omissions := leader_omissions +1;
20:           [ leader_omissions > Ot  →  W_suspect_silent(leader) ]
21:       □ leader ≠ W_leader() →
22:           leader := W_leader();
23:           leader_omissions := 0;
24:           [ leader = i →
25:             *[ En_attente ≠∅ →              % retransmission de messages qui ne
26:                 P-multicast(LAN, mσ ); ] ] ].        %  sont pas encore transmis.
```

Telque :

T*v* : Temps prévu pour voter un message;

O*t* : Seuil d'omission pour le leader;

T*f* : Temps prévue pour expédier un message;
leader : leader courant initialisée à 0;
leader_omissions : un conteur des omissions de leader initialisée à 0;
Vote : Messages étant votés initialisée à ∅;
En_attente : Messages pas encore expédiés initialisée à ∅;
Transmis : Messages expédiés avant leur arrivée initialisée à ∅.

■ **La sémantique de la commande alternative :**

Le comportement d'une commande alternative sécurisée en ce qui concerne l'état de sécurité de flux peut être décrit comme suit. Pour un état (de sécurité de flux) donné σ :

$$([\ i = 1..N \ \square\ C_i \rightarrow S_i\ ;], \sigma) \rightarrow (\text{update1}\ ;\ S_i\ ;\ \text{update2}, \sigma)$$

Où

update1 $\hat{=}$

$indirect := \{\ c \cup \overline{c} \mid c \in C_{bool}\ \}\ o\ indirect; \overline{l} := \overline{l} \cup \{\ c \cup \overline{c} \mid c \in C_{bool}\ \}\ \forall\ l \in lhs\_vars$

update2 $\hat{=}$

$indirect := tail(indirect)$

lhs_vars est l'ensemble de variables apparaissant du côté gauche de l'opérateur $:=$ dans les branches de la commande. C_{bool} est l'ensemble de variables apparaissant dans les *guards*. Naturellement, la branche exécutée dépend de l'état fonctionnel.

La transition est expliquée comme suit. Les flux indirects –à partir des guards de commande– existent seulement pendant le corps de la commande. Ainsi, *indirect* est mis à jour sur l'entrée (update1) avec la nouvelle valeur indirecte de flux qui est enlevée sur la sortie (update2). Comme les variables dans les branches non exécutées ne voient pas les effets d'*indirect*, toutes les variables qui peuvent recevoir des informations dans la commande, *lhs_vars*, ont leurs variables de sécurité mises à jour avec la valeur de flux des variables de *guard* sur l'entrée (update1).

Une règle décrivant la sémantique de la commande alternative est la suivante. On suppose $\overline{C_{bool}} = \{ c \cup \overline{c} \mid c \in C_{bool} \}$;

$R_{S-alternative}$

$$\frac{\begin{array}{c} \mathrm{P} \Rightarrow \mathrm{R}\,[\, indirect \leftarrow \overline{C_{bool}}\ \mathrm{o}\ indirect; \overline{l} \leftarrow \overline{l} \cup \overline{C_{bool}}\ \forall\ l \in lhs\_vars\,] \,, \\ \forall\ i = 1..N\ \ \{\ \mathrm{R}\ \}\ S_i\ \{\ \mathrm{T}\ \}\ , \\ \mathrm{T} \Rightarrow \mathrm{Q}\,[\ indirect \leftarrow tail(indirect)\ ] \end{array}}{\{\ \mathrm{P}\ \}\,[\ C_1 \rightarrow S_1 \square\ C_2 \rightarrow S_2 \square\ \ldots\ldots\ldots \square\ C_N \rightarrow S_N\,]\{\ \mathrm{Q}\ \}}$$

La règle est adaptée vers des preuves de sécurité de flux. Par conséquent, pour un prédicat R, il y a un prédicat T qui est établi après que la branche de commande S_i a exécuté. La relation entre P et R et entre T et Q est celle engendrés par la sémantique de transfert qui capture les modifications faites à l'état de sécurité de flux dans *update1* et *update2*.

Lemme 1 : dans l'algorithme au-dessus, si un message légal est reçu par une réplique correcte qu'elle n'est pas récupérée pour assez longtemps, il sera signé.

Preuve :

i) Quand un message légal *m* est reçu par une réplique correcte, il sera vérifié par le moteur de politique (ligne 1), stocké dans l'ensemble de *vote* (ligne 2) et puis passé au Wormhole pour être signé (ligne 3).

ii) Sachant que le service W_sign() du Wormhole retourne seulement un MAC s'il est appelé par au moins f +1 répliques.

iii) Par l'algorithme, si une réplique reste plus que T_V bloqué à la ligne 3 (i.e. le message m n'est pas encore signé par f+1 répliques correctes), la retransmission de la ligne 16 sera déclenchée périodiquement jusqu'à Vote =∅ (qui se produit seulement si la réplique exécute la ligne 4, après que le Wormhole signe le message). Puisque le multicast est juste, si une réplique retransmet m autant de fois, alors toutes les répliques correctes le recevront. Étant donné n ≥ 2 f +1, il y aura toujours f +1 répliques correctes qui recevront *m* et appelleront W_sign(m). Quand ceci se produit et de ii), le Wormhole produira la signature et la retournera au Payload.

Maintenant nous présentons des théorèmes pour les propriétés de validité et d'intégrité (cf. § **2.5.2.3**).

Théorème 1 (validité) : l'algorithme ATM s'assure qu'un message légal reçu par au moins une réplique CIS correcte qu'elle est récupérée récemment, est expédié à sa destination.

Preuve :

i) Par **Lemme 1**, nous savons qu'un message *m* reçu par une réplique correcte sera signé. Puisqu'un message est signé seulement si f+1 processus l'approuvaient, nous savons que au moins une réplique correcte qui a signé *m* le stockera dans l'ensemble d'attente 'En_attente'. Le message restera dans cet ensemble jusqu'à ce qu'il soit reçu dans le LAN, c.-à-d. il a été expédié (des lignes 10-12). Tandis qu'il est stocké dans cet ensemble *m* sera retransmis périodiquement (ligne 16). Après Ot retransmissions, on suspectera la réplique du leader courant (ligne 20).

ii) En raison de l'hypothèse de broadcaste juste, toutes les répliques correctes recevront *m* et l'approuveront, signeront et stockeront dans leurs ensembles d'attentes 'En_attente '.

iii) En conséquence, si le message n'était pas expédié par le leader l, alors f +1 répliques le suspecteront, et le Wormhole changera la réplique leader. Ce procédé de changement de leader peut se durer au plus f unité-temps jusqu'à ce qu'une réplique correcte devient leader, expédie *m* à sa destination, et le stocke dans son tampon d'attente ''En_attente buffer'' (lignes 21-26).

La notion d'une réplique étant récupéré récemment devrait maintenant être claire. Supposons que seulement une réplique correcte CISi reçoit le message. Dans ce cas, CISi ne doit pas être récupérée (on perd le message) jusqu'à que d'autres répliques correctes (qui sont également récupérées récemment) ont le reçu.

Théorème 2 (intégrité) : l'algorithme ATM s'assure qu'un message illégal n'est jamais traité par sa machine de destination.

Preuve :

i) dans ce modèle de système, la machine de destination ne peut pas être corrompu et elle traite seulement des messages signés avec la clé K ;

ii) la clé K est stockée dans le Wormhole, et un message m est seulement signé avec cette clef si au moins f +1 répliques CIS appellent W_sing(m).

iii) Étant donné ces deux faits, il est claire que des messages illégaux (approuvés au plus par f processus) ne peuvent pas être signés (en raison de (ii.)) et, ne soyez pas traité par leurs machines de destination même s'il est expédié par un leader malveillant (dû à (i.)). En conséquence, un message illégal ne sera jamais traité par sa machine de destination.

II) L'algorithme du service interactif (ASI)

Les services interactifs sont présentés dans l'algorithme ASI. La clé partagée K est employée comme paramètre dans quelques services. D'ailleurs, les services fournis par chaque wormhole local Wi emploient quatre variables : le *current_leader* stocke l'*identificateur* du *leader* courant ; *V* est un vecteur des ensembles tel que chaque ensemble V[σ] stocke les identificateurs des répliques qui approuvent actuellement un message avec MACσ ; et les ensembles Suspectm et suspects contiennent les processus qui suspectent la réplique i d'être, respectivement, malveillante ou silencieuse.

■■ Le Psuedo-code de l'algorithme ASI :

ASI :: [W_sign(m) || W-receive(x, y)]

W_sign(m) ::
1: σ := MAC(m,K); % admettre un contrôle de flux de la
2: W-multicast(W, ⟨VOTE, i,σ⟩); % commande repetitive.
3: *[|V[σ]| ≥ f +1 →
4: return σ ;] % retourner un mac valide dénoté σ.
W-receive(x, y)::
5: [receive(W, ⟨VOTE, j,σ⟩) →
6: V[σ] := V[σ] ∪ { j}; % enregistrer le vote de *j* sur un MAC*σ*.
7 : □ receive(j, ⟨SUSPECT-SILENT⟩) →
8 : $Suspect_s$:= $Suspect_s$ ∪ { j};
9 : □ receive(j, ⟨SUSPECT-MALICIOUS⟩) →
10: $Suspect_m$:= $Suspect_m$ ∪ { j}].

Telque :
K : la clé d'authentification (Service à clé symétrique) ;
current_leader : leader courant initialisée à n;
∀σ : V[σ] : Processus approuvant un message avec hachage h initialisée à ∅ ;
$Suspect_m$: Processus me suspectant d'être malveillant initialisée à ∅;
$Suspect_s$: Processus me suspectant d'être silencieux initialisée à ∅.

■ **La sémantique de la commande répétitive :**

Considérez ainsi le comportement de la commande réitérée concernant l'état de sécurité du flux :

$$(*[i = 1..N \square C_i \rightarrow S_i ;],\sigma) \rightarrow (\text{update1} ; S_i ; \text{update2}; (*[i = 1..N \square C_i \rightarrow S_i ;],\sigma)$$
$$\text{Ou } (\text{update3},\sigma)$$

Où update3 ≙
$indirect := \{ c \cup \overline{c} \mid c \in C_{bool} \} \uplus indirect; \overline{l} := \overline{l} \cup \{ c \cup \overline{c} \mid c \in C_{bool} \}$

Dans chaque itération de la boucle, la valeur de flux sur la *guard* de la boucle est empilée dans la pile *indirect* (**update1**) et dépilée à la fin (**update2**). Quand la boucle se termine, le flux indirect des conditions de la boucle à toutes les variables qui pourraient avoir reçu un flux dans la boucle (*lhs_vars*, pour couvrir le cas quand aucune branche ne s'exécute) et toutes les variables qui reçoivent par la suite un flux est enregistré (**update3**). Note comment l'opération $\uplus$ est employée au lieu du o pour capturer la permanence du changement sur *indirect*.

Une règle qu'exprime la façon dont la commande réitérée sécurisée influence l'état de sécurité de flux est :

R_S *–répétitive*

$$\frac{\begin{array}{c} P \Rightarrow R\ [\ indirect \leftarrow \overline{C_{bool}}\ o\ indirect;\ \overline{l} \leftarrow \overline{l} \cup \overline{C_{bool}}\ \forall\ l \in lhs\_vars\ ]\ , \\ \forall\ i = 1..N\ \ \{\ R\ \}\ S_i\ \{\ R\ \}\ , \\ R \Rightarrow P[\ \ indirect \leftarrow tail(indirect)\ ], \\ P \Rightarrow Q\ [\ indirect \leftarrow \overline{C_{bool}}\ \ \uplus\ indirect;\ \overline{l} \leftarrow \overline{l} \cup \overline{C_{bool}}\ \forall\ l \in lhs\_vars\ ] \end{array}}{\{\ P\ \}\ *[\ C_1 \rightarrow S_1\ \square\ C_2 \rightarrow S_2\ \square\ \ldots\ldots\ \square\ C_N \rightarrow S_N\ ]\ \{\ Q\ \}}$$

Nous pouvons justifier cette règle en utilisant la transition donnée ci-dessus. Car le corps de la commande peut être exécuté plusieurs fois, on a besoin d'un invariant sur l'état de sécurité de flux. ***P*** sert pour cet invariant. D'ailleurs, les modifications qui se produisent aux *lhs_vars* et aux variables *indirects* au début et à la fin de chaque branche permettent à un invariant intérieur ***R*** d'être établi. Après la terminaison, les modifications finales aux variables du flux établissent un état satisfaisant ***Q***.

Théorème 3 (signature) : L'algorithme ASI s'assure que le service W_sign(m) retourne un MAC produit avec la clé partagée K pour le message *m*, si et seulement si au moins f +1 répliques CIS l'appelaient en leurs Wormholes locaux.

Preuve :

i) Quand le service W_sign(m) est appelé à un Wormhole local, il se bloque jusqu'à la réception des messages de VOTE avec un MACσ correct par au moins f +1 Wormholes locaux (ligne 3).

ii) Un message de VOTE avec un MACσ correct est envoyé par le Wormhole local si et seulement si W_sign(m) s'appelle (lignes 1 et 2), puisque nous assumons que les MACs sont collision-résistants. En conséquence, le service W_sign(m) retourne un MAC produit avec la clé partagée K pour le message m, si et seulement si au moins f +1 répliques CIS l'appelaient en leurs Wormholes locaux.

III) L'algorithme du recouvrement Proactif-réactif (ARPR)

Le recouvrement Proactif-réactif lance des rétablissements périodique (temps-déclenché) et à chaque fois qu'un comportement malveillant est détecté ou suspecté

(événement-déclenché), l'algorithme ARPR représente une implémentation de ce service (Voir la stratégie **PRRW.** cf. § **2.5.2.2**).

■■ Le Pseudo-code de l'algorithme ARPR :

ARPR :: [Proactive_recovery() || Réactive_recovery()]

```
Proactive_recovery() ::                                  % admettre un contrôle de flux de la
1: synchronize_global_clock();                                    %  commande repetitive.
2: t_last := global_clock()−T_P+((i−1)T_slot + f T_D);
3: *[
4:      wait until global_clock() = t_last +T_P;          % la prochaine période.
5:      t_last := global_clock();
6:      recover();                                        % récupération de k répliques.
7:  ]
Réactive_recovery() ::                                   % admettre un contrôle de flux de la
8:   *[ |Suspect_m| ≥ f +1 →                                      %  commande repetitive.
9:          recover();                                    % récupération d'une réplique.
10:     □ |Suspect_m| < f +1 ∧ |Suspect_s+Suspect_m| ≥ f +1 →
11:          ⟨j, p⟩ := allocate_subslot();
12:     [ j ≠ i →
13:        wait until global_clock() mod T_P = ( j−1)T_slot +(p−1)T_D;
14:        recover();] ]                                 % récupération d'une réplique.
```

Tel que :

T_P : Période de rétablissement;

T_D : durée de rétablissement d'une réplique;

$T_{slot} = (f +1)T_D$: durée de Slot ;

T_{last} : l'instant du dernier recouvrement périodique initialisé à 0.

Théorème 4 (nombre de répliques récupérées) : L'algorithme ARPR assurent qu'au plus une réplique correcte est en récupération à tout moment.

Preuve :

Considérons une réplique ***i,*** un rétablissement peut seulement être déclenché à une des trois raisons suivantes :

1. Rétablissement proactif périodique (ligne 6) ;
2. Rétablissement réactif parce que la réplique ***i*** est compromise (ligne 9) ;

3. Rétablissement réactif parce que la reproduction ***i*** est suspectée pour être silencieuse (ligne 14).

Nous devons prouver que si la réplique i est récupérée due à la raison 1 ou 3, alors elle est garantie qu'aucune autre réplique correcte n'est pas en récupération en même temps. Si la réplique i a été récupérée à cause de la raison 2 alors elle est compromise, et ainsi rien ne doit être prouvé.

i) Si la reproduction i a été récupéré à un rétablissement proactif périodique (raison 1), alors l'algorithme proactif de rétablissement (lignes 1–7) garantit que la réplique i commence les rétablissements toutes les T_{slot} unités-temps après le rétablissement périodique précédent. Étant donné que $T_{slot} \geq T_D$ et que T_D est la durée maximum de rétablissement, alors on assure que quand une réplique i commence un recouvrement périodique, le rétablissement périodique précédent a déjà fini. D'ailleurs, l'algorithme de rétablissement réactif (lignes 8–14) garantit que la réplique i toujours commence les rétablissements au moins en T_D unités-temps après le rétablissement réactif précédent. Par conséquent, il est assuré que quand une réplique i commence un rétablissement proactif, les rétablissements proactifs et réactifs précédents ont déjà fini.

i) Si la réplique ***i*** a été récupéré à cause d'un rétablissement réactif parce qu'elle est suspecté d'être silencieux (raison 3), alors l'algorithme de rétablissement réactif (lignes 8–14) garantit que la réplique i commence toujours des rétablissements au moins T_D unités-temps après le rétablissement périodique précédent. Étant donné que T_D est le temps maximum de rétablissement, alors il est assuré que quand une réplique i commence un rétablissement réactif de cette nature, le rétablissement périodique précédent a déjà fini.

Théorème 5 : Le service de protection CIS est épuisement-safe (c.-à-d., au plus f répliques sont échoués en même temps).

Preuve :

Suivant les indications de [*SOU06*], s'il est possible de limité le temps nécessaire pour qu'un attaquant peut produire f +1 échecs dans le système pendant un T_{exhmin} constant, alors il est garanti que pas plus de f répliques ne seront défectueux en même temps, tant que $T_P + T_D < T_{exhmin}$.

5.2.1.2 Un système de communication sécurisé pour les ICs

Afin d'assurer la confidentialité des données échangée entre une application distante et une station de l'IC (Infrastructure Critique), nous allons appliquer, un contrôle de flux sur les communications effectuées (sous forme de messages) du WAN au LAN et l'inverse, en montrant la fiabilité d'un tel contrôle.

Nous considérons que l'administrateur de système ait assez d'information pour prendre des décisions concernant l'utilisation de ressource, ainsi chaque réplique du système Cis-firewall maintient un journal de qui envoient les messages à qui. Pour garantir la confidentialité des données de l'IC, les répliques du Cis-firewall ne doivent pas pouvoir lire l'information critique contenue dans chaque message qui les traverse.

Comme nous avons déjà énoncé, nous pouvons décrire la globalité de système à base d'IC en utilisant le langage CSP :

IC :: [Station_send[i] || Cis-firewall[i] || Station_receive[i]]

Tel que,

IC : Le programme principal qui modélise l'infrastructure critique,

Station_send[i] : Un processus (application) expéditeur du WAN ou du LAN,

Station_receive[i] : Un processus récepteur du WAN distant ou du LAN,

Cis-firewall[i] : Le pare-feu Cis interceptant chaque message du WAN au LAN et l'inverse, tel que :

Cis-firewall[i] :: [ATM || ASI || ARPR], (Trois algorithmes qui s'exécutent en parallèle, voir figure **5.2**).

Station_send[i] ::	**Station_receive[i] ::**
Var dest, smsg ;	**Var** source, cmsg ;
*[IApp **?** dest →	*[Cis_i **?** source →
IApp **?** smsg ;	Cis_i **?** cmsg ;
Cis_i **!** dest ;	IApp **!** cmsg ;
Cis_i **!** smsg ;]	IApp **!** source ;]

Afin de bien illustrer le contrôle de flux et ses démonstrations de sûreté, on prend une version simplifiée de l'algorithme ATM soit ''Cis_i'':

Cis_i ::
Var message, dest, i, req ;
Var no_comms ; **array** send_comms, dest_comms;
Message := 0; no_comms := 0 ; i := 0 ; dest := 0 ; req := 0;
*[Admin **?** req → Admin **!** send_comms ; Admin **!** dest_comms ;
□ Comms → Station_send[i] **?** dest;
Station_send[i] **?** message ;
no_comms := no_comms +1;
send_comms[no_comms] := i;
dest_comms[no_comms] := dest;
Station_receive[dest] **!** i ;
Station_receive[dest] **!** message ;]

Des deux côtés WAN et LAN, le système SCADA contient plusieurs processus, de type Station_send et Station_receive, pour envoyer et recevoir des messages traversant le pare-feu Cis. Le Cis-firewall accepte les messages des applications communicantes, enregistre l'expéditeur et le récepteur et puis transmet le message. Le Cis-firewall accepte également des requêtes d'un processus Admin (non montré) pour des informations concernant l'utilisation du pare-feu.

■ Exigences et preuve de sécurité : Le système est sécurisé si le Payload ne peut pas connaître le texte des messages critiques qui traversent le service. Ainsi, les données passées à l'administrateur, send_comms et dest_comms, ne doivent pas avoir reçu un flux d'information à partir des variables de message d'aucun processus Station_send.

Exig-Séc $\hat{=}$ Station_send[i].message $\notin \overline{send\_comms} \cup \overline{dest\_comms} \cup$ val(*indirect*)

Est toujours vrai dans la réplique Cis_i, c'est-à-dire, l'invariant **P** de la boucle de Cis_i assure **Exig-Séc**. Nous choisissons **P** comme suit (extrait de la première itération) :

P $\hat{=}$ { {cs_dest} $\subseteq \overline{dest}$, {cs_msg} $\subseteq \overline{message}$, {no_comms} $\subseteq \overline{no\_comms}$, {send_comms} $\subseteq \overline{send\_comms}$, {dest_comms} $\subseteq \overline{dest\_comms}$, {dest, no_comms, dest_comms, cs_dest} $\subseteq \overline{dest}$, {Comms} $\in$ *indirect*}

Car **Exig-Séc** ⇒ **P**, nous devons juste prouver que **P** est en effet l'invariant, et pré-condition, de la boucle du Cisi, cs_dest dénote la variable destination du processus Station_send. Les transferts initiaux dans le Payload établissent un état satisfaisant :

$$pre = \{ \overline{all} = \{\}, indirect = \prec \quad \{\} \quad \succ \quad \}$$

Où *all* représente chacune des variables du processus. L'axiome d'assignation est employé pour vérifier ceci. Par exemple, $\{pre'\}$ req := 0 $\{pre\}$ se tient, puisque $A_{s\text{-}:=}$ signifie que la pré-condition se tient si et seulement si :

$$pre' \Rightarrow pre[\overline{req} \leftarrow \text{val}(indirect)]$$

la valeur de val(*indirect*) est {}, alors la pré-condition pre' donnée par l'axiome est pre avec le sous-prédicat pour $\overline{req}$ enlevé. Suivant ce raisonnement, pour chaque commande, le prédicat parvenu par la pré-condition du Payload est val(*indirect*)= {} ; ceci doit être vrai car *indirect* de chaque processus est initialement vide. Le résultat est qu'on peut choisir **P** une pré-condition (de la boucle) satisfaisante; nous devons maintenant prouver seulement que **P** est également un invariant de la boucle.
Il y a deux branches dans la commande réitérée, la première gardée par des commandes de communication (la liste de commandes de cette branche se compose seulement par des commandes de communications). Selon la règle réitérée et les axiomes pour les commandes de communications, nous pouvons considérer **P** être la post-condition de cet ordonnancement des commandes (puisque n'importe quelle post-condition peut être choisie). Et que **P** est une post-condition correcte sera prouvée dans la deuxième phase de la preuve.

La deuxième branche de la commande réitérée est gardée par une expression booléenne. De la règle réitérée, nous définissons un invariant interne **R** comme suit :

$$R = \text{P} \text{ telque } indirect = \prec \{\text{Comms}\} \{\} \succ$$

R est le même prédicat que P sauf que *indirect* est $\prec$ {Comms} {} $\succ$. en utilisant le fait que $\overline{C_{bool}}$ = {Comms}. Les deux dernières lignes de la liste des commandes de cette branche sont des commandes de communication et nous choisissons R en tant que leur pré-condition, supposant que **R** est une post-condition des commandes

restantes, qui serait vrai si R était l'invariant interne. La dernière commande de la liste restante est l'affectation dest_comms[no_comms] := dest. **R** est la post-condition, nous prenons R' une pré-condition. À partir de $A_{s\text{-}:=}$, R' est définit comme :

$$R[\overline{dest\_comms} \leftarrow \{\text{dest\_comms, no\_comms, dest}\} \cup \overline{dest\_comms} \cup \overline{dest} \cup \overline{no\_comms} \cup \text{val}(indirect)\ ]$$

$$R[\overline{dest\_comms} \leftarrow \{\text{dest\_comms, no\_comms, dest, c\_msg}\}\ ]$$

R **mais** dest_comms est **enlevée.**

Le prédicat **P** mais v est enlevée, est le même prédicat **P** mais sans aucun sous-prédicat dans la variable v. On suit ce chemin pour les deux autres commandes, la pré-condition sur la commande $no\_comms := no\_comms + 1$, nous prenons :

R **mais** dest_comms, send_comms sont **enlevées**

Comme $R' \Rightarrow R$ mais dest_comms, send_comms enlevées, nous insérons R comme une pré-condition en utilisant la règle standard de conséquence. Enfin puisque les deux premières commandes de la liste de commandes de la branche sont des commandes de communication, nous postulons que R est une post-condition appropriée.

Ainsi R est préservé par la deuxième branche, c'est un invariant interne. Puisque R a été dérivé de P en utilisant $R_{s\text{-}répétition}$, P est en effet l'invariant de la boucle et ainsi le processus de Payload est prouvé à condition que toutes les hypothèses faites au sujet des commandes de communication soient vraies.

Que ces hypothèses sont vraies peut être prouvé dans la deuxième étape de la preuve ce qui suit maintenant :

Les processus ''*Station*'' ont seulement des commandes de communication dans leurs corps de boucle. Dans les processus Station_send, nous choisissons val(indirect) $\subseteq$ {Comms} en tant que l'invariant de la boucle.

Nous montrons maintenant que les preuves des processus coopèrent. Prenons la communication (Station_send[i]) || (Cisi ! smsg) comme un exemple. L'axiome de communication exige que les post-conditions suivantes se tiennent :

$\overline{\text{message}} = \{\text{smsg}\} \cup \text{val}(\text{indirect}_{Cisi}) \cup \text{val}(\text{indirect}_{Station\text{-}send}) \subseteq \{\text{Comms, smsg}\}$

$\text{indirect}_{Cisi} \subseteq \{\text{Comms}\}$

$\text{indirect}_{Station\text{-}send} \subseteq \{\text{Comms}\}$

Ce qui est vérifié dans les post-conditions que nous avons choisi. La preuve des

autres communications est semblable. En prouvant que les preuves des processus coopèrent, nous pouvons être satisfaits que les hypothèses faites dans la première étape de la preuve du processus Payload sont valides, et ainsi que l'invariant de la boucle est vraie. Puisque cet invariant répond aux exigences de sécurité, le pare-feu Cis_firewall est prouvé sécurisé.

Ainsi un attaquant dissimulé en prenant le chemin d'une approbation malveillante ou d'un suspect malveillant ne peut pas dévoiler les données sensibles de l'IC, en effet, l'application d'un tel contrôle empêche la divulgation de contenus des messages passants à travers le pare-feu Cis-firewall.

5.2.2 Le module de la protection interne

Dans ce qui suit, nous allons présenter une amélioration de modèle du contrôle de flux d'information en montrant de nouvelles définitions et axiomes avec des démonstrations de sûreté et de nouvelles règles. Les règles de contrôle de flux que nous avons proposées sont utilisées pour réduire le champ de vulnérabilité touchant les systèmes d'infrastructure critique (i.e. SCADA, PCS[1], etc.) notamment la protection de l'IC contre les menaces internes (scénario 2 : en empêchant l'induction des informations non autorisées).
La figure **5.3** montre le niveau de contrôle de flux en appliquant une stratégie pour prévenir l'induction des informations dans une sous-station de l'infrastructure critique.

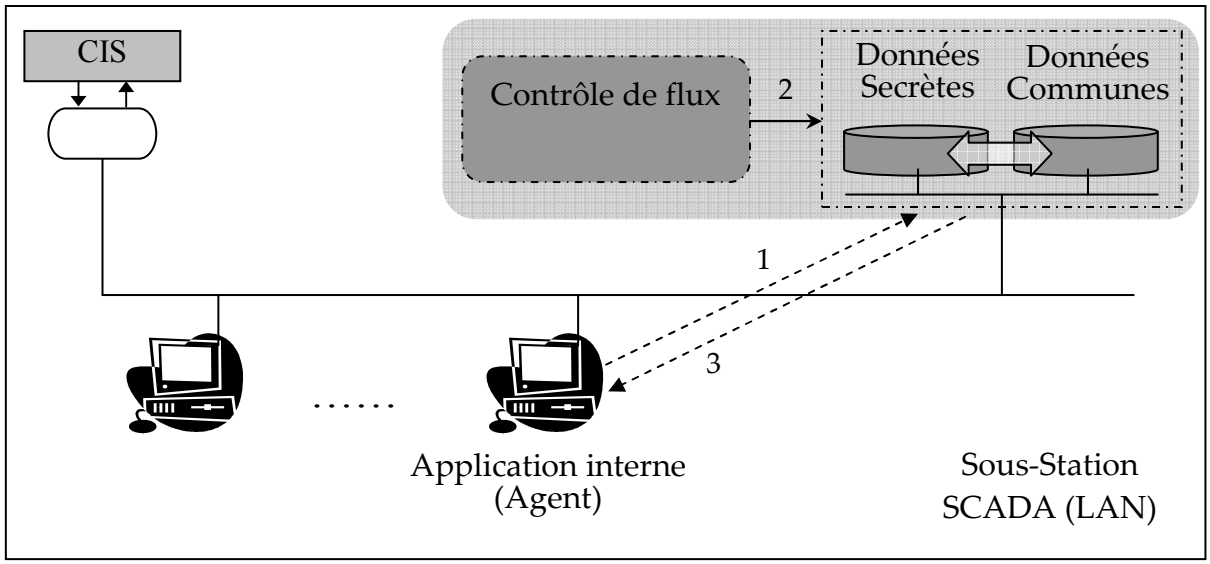


Figure 5.3 : Le contrôle de flux sur une base d'informations secrètes

[1] PCS : Process Control System

5.2.2.1 Nouvelles règles et leur sémantique pour le CFI

Quand nous voulons analyser correctement des problèmes que nous devons résoudre, il est important de revenir au concept de la politique de sécurité pour la confidentialité. Dans cette extension, nous considérons seulement le cas du contrôle d'accès basé sur le concept d'organisation (en respectant la politique adoptée au Cis-firewall servant à la protection externe). Une organisation définit une politique OrBAC qui est appliquée à un ensemble de formules (relations) **L** qui représente le domaine de la connaissance (base d'informations relationnelle) de l'organisation et un ensemble des agents **Γ** membres de cette organisation. Pour chaque agent $A \in \mathbf{\Gamma}$, la nouvelle politique divise l'ensemble de formules (schémas) **L** en deux sous-ensembles :

1. L'ensemble de formules (schémas) C_A pour lesquelles A est explicitement autorisé d'avoir un accès.
2. L'ensemble de formules P_A pour lesquelles on interdit explicitement A d'avoir un accès.

Nous supposons que la politique est complète. Ceci signifie que chaque formule de **L** doit appartenir à C_A ou à P_A. Par exemple, prenons le modèle de la politique de sécurité à multi-niveaux. Quelques formules $s \in \mathbf{L}$ reçoit une classification l_s et chaque agent $A \in \mathbf{\Gamma}$ reçoit une autorisation L_A. Les ensembles C_A et P_A sont alors définis par :

$$C_A = \{s \in \mathbf{L} \mid l_s \leq L_A\},\ P_A = \{s \in \mathbf{L} \mid \neg (l_s \leq L_A)\}$$

Nous rappelons que le problème d'induction dans les bases de données à multi-niveaux peut être défini par ce qui suit : un agent ***A*** peut dériver l'information sensible de l'information non-sensible à laquelle ***A*** a légalement accès. Il semble a priori facile de résoudre ce problème parce que nous pouvons croire qu'il est suffisant de diviser arbitrairement l'ensemble de relations appropriées **L** en deux ensemble : C_A des permissions explicites pour ***A***, et P_A des prohibitions explicites pour ***A***.

En effet, le problème est beaucoup plus pénible parce que chaque agent A peut employer sa propre permission de savoir l'information mais également des règles générales et des connaissances communes (code des applications, des protocoles,

etc.) pour dériver de nouvelles informations (qui sont probablement interdites). Ainsi, nous devons vérifier, que les ensembles C_A et P_A ont été définis d'une façon cohérente.

Afin d'analyser correctement ce problème, nous avons besoin d'une définition formelle des concepts suivants :

1) La permission de savoir de chaque agent ''*PS*'' : On considère, qu'un agent A est autorisé pour connaître chaque information donnée α pour laquelle A est explicitement autorisé d'avoir un accès, et nous proposons une sémantique formelle pour ce concept :

$$[\ \alpha \in C_A \rightarrow V_A := V_\alpha\ ] \equiv \quad \textbf{Si}\ \alpha \in C_A\ \textbf{alors}\ PS_{A\alpha}$$

V_A : une variable (structure) du processus (agent) A qui va recevoir l'information α.

Pour assurer la confidentialité de l'information α, pendant l'affectation, la règle d'assignation est appliqué :
On prend la commande $V_A := V_\alpha$, cette commande affecte la variable de sécurité $\overline{V_A}$ à :

$$\{\ V_\alpha\ \} \cup \{\ \overline{V_\alpha}\ \} \cup val(indirect)$$

Le terme $\{V_\alpha\}$ capture le flux d'information direct à partir de chaque composante de la variable V_α. Le terme $\{\ \overline{V_\alpha}\ \}$ capture la transitivité dans les flux d'information, *val*(*indirect*) sauvegarde la valeur du flux que la variable de côté gauche recevra indirectement.

Quand V_A est remis à zéro ou prend une affectation n'impliquant pas α, alors α n'est plus un membre de la variable de sécurité $\overline{V_A}$ puisque le nouveau contenu de l'information du V_A est indépendant de n'importe quelle valeur précédente de V_α.

L'effet de l'affectation sur l'état de la sécurité de l'application est capturé par l'axiome suivant :

$$\{\ P[\ \overline{V_A} \leftarrow (\ \{\ V_\alpha\ \} \cup \{\ \overline{V_\alpha}\ \} \cup val(indirect)\ )]\ \}$$
$$V_A := V_\alpha$$
$$\{\ P\ \}$$

D'ailleurs, A est implicitement autorisé pour réaliser des dérivations valides, c.-à-d. :

$$\textbf{Si } PS_{A\alpha} \textbf{ et } PS_{A(\alpha \rightarrow \beta)} \textbf{ alors } PS_{A\beta}$$

Cette règle veut dire : si A est autorisé de savoir α et si A est autorisé de savoir que α mène à β (α → β), alors A est autorisé à effectuer la dérivation. Par conséquent, A est autorisé de savoir β.

2) La prohibition ''Interdiction'' de savoir de chaque agent : nous le dénotons ''*IS*''. Nous devons formellement définir ce concept. Comme dans le cas de la permission de savoir, la sémantique de IS_A doit imposer qu'on interdise un agent A de savoir n'importe quelle information donnée α pour laquelle A est explicitement interdit d'avoir un accès, c.-à-d. :

$$[\ \alpha \in P_A \rightarrow V(A) :\!= \text{NIL}\ ] \equiv \textbf{Si } \alpha \in P_A \textbf{ alors } IS_A\alpha$$

D'ailleurs, il existe des prohibitions implicites, par exemple :

$$\textbf{Si } IS_{A\alpha} \textbf{ ou } IS_{A\beta} \textbf{ alors } IS_{A(\alpha \wedge \beta)}$$

Cet axiome veut dire : si A est interdit de savoir α ou si A est interdit de savoir β alors on interdit implicitement A de savoir la conjonction $\beta \wedge \alpha$.
La phrase suivante énonce une autre prohibition implicite :

$$\textbf{Si } IS_{A(\alpha \vee \beta)} \textbf{ alors } IS_{A\alpha} \textbf{ et } IS_{A\beta}$$

Cela a pu être lu : si on interdit A de savoir les données disjonctives $\alpha \vee \beta$, alors on interdit également implicitement A de savoir des données plus instructives telles que β ou α.
Ce dernier axiome nous permet de contrôler une divulgation partielle d'information. Par exemple, laissez-nous supposent que le mot de passe d'un agent secret B est ''monde'' (soit : 6D.6F.6E.64.65 en hexadécimal, 470020940901 en décimal) et que l'administrateur de sécurité indique réellement que l'agent non classifié A est interdit de savoir que ce mot de passe est entre 4.10^{11} et 5.10^{11} c'est-à-dire :

$$IS_A\ (\text{Password}(B, 4.10^{11}) \vee \ldots \vee \text{Password}(B, 5.10^{11}))$$

Par conséquent, en employant l'axiome ci-dessus, on interdit également implicitement A de réduire l'ensemble de valeurs appartenant à l'intervalle [4.10^{11}, 5.10^{11}].
D'une part, nous considérons l'axiome suivant :

$$\textbf{Si} \quad IS_{A\,\alpha(c)} \quad \textbf{alors} \quad \neg\,(\,IS_{A\,(\exists\,x,\,y)}\,)$$

Où $y \equiv IS_{A\,\alpha(x)}$ et $\underline{c}$ est une constante donnée.
Cet axiome signifie que : si A est interdit de connaître une information secrète α(c) (par exemple Destination(Entreprise, c)), alors on ne devrait pas implicitement interdire A de savoir l'existence de cette information secrète. Dans notre approche, c'est le rôle de la politique de sécurité d'indiquer explicitement le cas pour lequel l'existence d'une information secrète est également secrète.
Notez également, que nous devons imposer la validité de la formule suivante :

$$\neg\,(PS_{A\alpha} \wedge IS_{A\alpha})$$

Cette règle peut être interprétée comme suit : ***A*** ne peut pas avoir la permission de connaître α et la prohibition de connaître α. L'application de cette condition garantit que ***A*** ne peut pas déduire l'information interdite de l'information autorisée en réalisant des dérivations valides. Cependant, quelques difficultés demeurent parce que ***A*** peut également réaliser des dérivations plausibles.

5.2.2.2 L'obscurité pour remédier aux dérivations plausibles

On reprend l'exemple du chapitre **04** (scénario **2**). Quand des requêtes sont envoyées au système de gestion de bases de données qui contrôle la relation SDT décrite dans le tableau **5.1,** les réponses qui sont formulées par ce système de gestion de bases de données, et selon notre approche, peuvent dépendre de plusieurs paramètres :

Starship	Destination	Type-entrée
Enterprise	Rigel	***IN*** (secrète)
Enterprise	Restricted	***IN'***
Intergalactic	Talos	***IN'***

Tableau 5.1 : La relation à multi-niveau SDT

1. Le niveau d'autorisation de l'agent qui envoie la requête au système de gestion de bases de données (c.-à-d. non classifié ou secret).
2. La "caractéristique" de la politique de sécurité qui est appliquée pour assurer les contrôles de flux internes de l'information et, probablement, les contrôles d'induction.
3. La méthode qui est employée pour éviter de telles inductions (c.-à-d. avec ou sans obscurité).
Il peut être ainsi intéressant de regarder le problème d'induction dans ces différents contextes, afin de pouvoir définir quelques solutions possibles pour résoudre ce problème. Considérons que la politique de sécurité est définie de sorte que l'Agent_u (agent non-classifié) soit interdit de savoir n'importe quelle information secrète et soit également interdit de savoir l'existence d'une telle information secrète. La solution est d'imposer l'uniformité de la politique de sécurité est de considérer que la base de données est complète et obscurcie (suivant les indications de Tableau **5.2**) :

$$\forall \text{ Star } [\exists \text{ dest, SDTN(star, dest)} \in \text{DS}] \Rightarrow$$
$$[\exists \text{ dest', SDTN(star, dest')} \in \text{DC} \wedge (\text{dest'} \neq \text{dest})]$$

Starship	Destination	Type-entrée	NC
Enterprise U	Rigel S	***IN*** (secrète)	S
Enterprise U	Talos U	***IN'***	U
Intergalactic U	Talos U	***IN'***	U

Tableau 5.2 : La relation obscurcie à multi-niveaux SDTN

Ceci signifie que chaque fait qui existe dans DS doit également exister dans DC, avec une valeur distincte pour chaque attribut. Considérons que la politique de sécurité est définie de sorte que l'Agent_u soit interdit de savoir n'importe quelle information secrète et soit également interdit de savoir l'existence d'une telle information secrète c.-à-d. :

$$IS_{\text{Agentu}} [\text{Destination (Enterprise, Rigel)}] \wedge$$
$$IS_{\text{Agentu}} [\exists \text{ dest}, IS_{\text{Agentu}} [\text{Destination (Enterprise, dest)}]]$$

En effet, il est possible de décomposer une relation à multi-niveaux en un ensemble de relations à niveau unique. On peut considérer ainsi que les deux bases de données distinctes (DS et DC : Figure **5.3**) sont gérées par le SGBD sécurisé pour représenter la base de données originale :

- DS contient les données secrètes de la base d'information auxquelles seulement les agents secrets, $Agent_S$, ont l'accès ;
- DC contient les données non classifiées (Communes) de la base auxquelles tous les $agent_S$ et n'importe quel agent non classifié, $Agent_u$, ont l'accès ;

■■ Le Pseudo-code de l'algorithme d'obscurité ''AO'' :

```
AO ::
Var R1, R2 ;
L(x) := ∅ ;
  *[ (∀ Ti ∈ R1 ∧ 1 ≤ i ≤ | R1 |) →          % | R1 | : nombre de tuples de R1
    L(x).Insert (Ti) ;
       [ Ti ∈ DS → K := Ti ;
         K.val(attributs) := l ;               % l : valeur commune ∈ DC
         L(x).Insert (K) ;]
     □ i > | R1 | → R2 := L(x);
  ]
```

L'application de cet algorithme sur le schéma SDT nous donne un nouveau schéma SDTN.

Soient deux processus $Agent_u$ (un agent non classifié qui, éventuellement, va envoyer des requêtes vers une base d'information) et Queryrun (exécutant de requêtes) :

$Agent_u$ ::	**Queryrun ::**
Var req, SQL ; *[Console **?** req → Console **?** SQL ; Queryrun **!** SQL ;]	**Var** ; tampBD, req, rep ; *[tampBD **?** req → tampBD **?** SQL ; rep := run(SQL) ; $Agent_u$ **!** rep ;]

Comme il a été montré dans le chapitre précédent, une requête de l'$Agent_u$ vers la table SDT :

SQL SELECT ⟨Destination⟩ FROM SDT
WHERE Starship = Enterprise

Retourne la $réponse_{u1}$: ⟨Restricted⟩

Par contre, la réponse envoyée par Queryrun à l'$Agent_u$ pour la même requête, mais sur la nouvelle table SDTN :

SQL SELECT ⟨Destination⟩ FROM SDTN
WHERE Starship = Enterprise

Retournera la $réponse_{u2}$: ⟨Talos⟩

Du fait que la base de donnée est obscurcie, l'$Agent_u$ peut seulement être sûr que :

$$\exists \text{ dest, Destination (Enterprise, dest)} \in DC$$

Grâce à cette nouvelle approche un agent interne d'un système SCADA ne peut pas exploiter des vulnérabilités découvertes en poursuivant l'induction basée sur l'élimination de possibilités décrite dans le chapitre précédant (i.e. Scenario 2).

5.3 Conclusion

Le modèle AI3S permet la prévention des données d'un système à base de l'information critique, il assure leur confidentialité et leur intégrité grâce à l'application des règles de flux d'information à chaque communication entre les sites de système.

A travers ce chapitre, nous avons défini formellement notre modèle AI3S pour la sécurité des systèmes d'infrastructures critiques (SCADA, PCS, etc.), nous avons développé le module responsable de la protection externe et nous avons proposé des algorithmes (écrits en CSP en associant à chaque commande un contrôle de flux approprié) permettant l'implémentation de ses fonctionnalités. Nous avons également développé le module responsable de la protection interne, ainsi nous avons donné de nouvelles définitions et règles de contrôle de flux en étendant le modèle original de contrôle de flux d'information afin d'augmenter la résilience du système contre les tentatives malveillantes.

Conclusion & Perspectives

Si un système (sûreté-critiques, i.e. les infrastructures critiques, SCADA, PLC; ou argent-critiques, i.e. les opérations bancaires en ligne, les sites Web du commerce électronique, etc.) doit être déployé dans un environnement (par exemple, Internet) où des attaquants malveillant peuvent être présents, alors il est certainement ne pas sûr de supposer que les menaces seront toujours identiques. En effet, il est bien connu que des attaquants sont activement engagés dans le développement de nouvelles techniques pour injecter ou activer les défauts latents dans les systèmes existants. La manière habituelle de combattre un tel comportement malveillant est d'appliquer des pièces correctrices de sécurité aux systèmes d'exploitation ou en fournissant de nouvelles (meilleures) versions du code d'application. Typiquement, ces activités sont faites manuellement par un administrateur de système et peuvent présenter des périodes d'indisponibilité dans le fonctionnement du système.
Nous concluons du fait que les systèmes argent-critiques et sûreté-critiques ne devraient pas dépendre de l'intervention humaine et que l'indisponibilité devrait être évitée à tout prix. En effet et dans ce travail, notre souci principal est avec les systèmes sûreté-critiques, étant donné la variation récente dans l'attention des attaquants des cibles argent-critiques classiques aux infrastructures sûreté-critiques (par exemple : énergie, eau, gaz, etc.).

Dans ce contexte, nous avons présenté un travail qui se concentre sur deux aspects de sécurité à savoir le contrôle de flux d'information et la tolérance d'intrusions dans les systèmes d'infrastructures critiques (i.e. SCADA, PCS, etc.). Notre modèle est développé en se basant sur le modèle proposé par P. Sousa, A. N.

Bessani, M. Correia, N. F. Neves, et P. Verissimo pour la protection des systèmes critiques en se basant sur la technique de tolérance d'intrusion dans les environnements distribués. Notre contribution à ce modèle consiste à augmenter sa résilience contre les attaques malveillantes en réduisant de ce fait ses points de vulnérabilités. Et de l'intégrer avec un modèle de contrôle de flux d'information, qui est à son rôle reçu une vérification et une extension pour assurer la confidentialité des données en plus de l'intégrité. Ainsi, notre solution fournit un modèle hybride (AI3S) pour la sécurité des données sensibles au sein des environnements critiques en intégrant un modèle de tolérance d'intrusions avec un modèle de contrôle de flux d'information.

Le modèle de tolérance d'intrusions est utilisé de telle sorte à protéger les sous-réseaux des systèmes SCADAs (comme un pare-feu) des menaces venant des réseaux ouverts comme l'internet, après vérification, nous avons découvert quelques vulnérabilités que nous avons corrigé en appliquant un contrôle de flux sur ce modèle. De sa nouvelle version, le modèle de contrôle de flux que nous avons proposé dans ce mémoire permet de contrôler les flux de communication en assurant, en plus de l'intégrité, la confidentialité des informations échangées entre les sites de l'infrastructure critique. La confidentialité est assurée en proposant une extension pour la non-induction des informations non autorisées. De ce fait, nous avons donné de nouvelles définitions, règles et leur sémantique pour pallier les lacunes du modèle original.

Dans les travaux ultérieurs, nous comptons étendre cette approche en étudiant d'autres problèmes liés à la gestion et la collaboration entre les organisations de l'IC (Infrastructure Critique) grâce à la technologie des services Web, tout en contrôlant que les interactions entre ces organisations sont conformes à leurs attentes et à leurs politiques internes spécifiées grâce à OrBAC[1], qui convient à un contexte multi organisationnel, il est facilement applicable à un environnement à grande échelle (plus particulièrement aux IC) car il n'impose aucune limitation sur le nombre ou la taille des organisations (passage à l'échelle), et il peut définir une politique de sécurité extensible et flexible (gérant les changements dans les organisations). L'architecture étendue offre une gestion décentralisée des politiques de contrôle d'accès où les organisations négocient mutuellement les contrats pour la collaboration. Le couplage entre les organisations est faible (loose coupling), et chaque organisation contient ses propres ressources, services, applications, système d'exploitation, règles de fonctionnement, objectifs et règles de politique

[1] Chaque réplique est dotée de ce type de contrôle d'accès.

de sécurité (spécifiées selon OrBAC). La technologie de services Web permet la communication entre des organisations sans connaissance approfondie des systèmes propres à chacune. Il n'est pas nécessaire de connaître la structure hiérarchique des autres organisations, on assure ainsi la confidentialité des données et de la façon selon laquelle les services sont implémentés.
L'extensibilité et la facilité d'utilisations des WS et d'OrBAC facilitent la gestion et l'intégration de nouvelles organisations (avec leurs utilisateurs, données, services, politique, etc.).

Bibliographie

[**ADE02**] Adelsbach, A., Alessandri, D., Cachin, C., Creese, S., Deswarte, Y., Kursawe, K., Laprie, J.C., Powell, D., Randell, B., Riordan, J., Ryan, P., Simmonds, W., Stroud, R., Verissimo, P., Waidner, M., Wespi, A.: Conceptual Model and Architecture of MAFTIA. *Project MAFTIA IST-999-11583 deliverable D21. (2002).*

[**AEK03**] A. Abou El Kalam, R. E. Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miège, C. Saurel, and G. Trouessin. Organization Based Access Control. In Proc. *of 4th IEEE Int. Workshop on Policies for Distributed Systems and Networks – Policy'03, June 2003.*

[**AMI00**] Amin, M. "National Infrastructures as Complex Interactive Networks." In Automation, Control and Complexity: An Integrated Approach, *pp. 263–286. New York: John Wiley & Sons, 2000.*

[**API04**] American Petroleum Institute. SCADA Security (API Standard 1164, Draft), *Appendix B. March, 2004.*

[**APT80**] Apt (K.R.), Francez (N.), De Roever (W.P.), A Proof System for communicating Sequential Processes, *in ACM Transactions on Programming Languages and Systems, volume 2 (3), July 1980, pages 359-385.*

[**AVI01**] Avizienis, A., Laprie, J.C., Randell, B.: Fundamental concepts of dependability. *Technical Report 01145, LAAS-CNRS, Toulouse, France (2001).*

[**BAN07**] M. Banatre, A. Pataricza, A. Moorsel, P. Palanque, and L. Strigini. From resilience-building to resilience-scaling technologies: Directions – ReSISTNoE Deliverable D13. *DI/FCUL TR 07–28, Dep. Of Informatics, Univ. of Lisbon, Nov. 2007.*

[**BEL98**] Bellovin, S. Cryptography and the Internet, Advances in Cryptology. *CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, CA: August 1998.*

[**BEL99**] S. M. Bellovin. Distributed firewalls. ; *November 1999.*

[**BEL03**] A. Belokosztolszki, D.M. Eyers, K. Moody. Policy contexts: Controlling information flow in parameterized RBAC. *IEEE 4th International Workshop on policies for Distributed systems and Networks, June 2003.*

[**BER97**] E. Bertino, P. Samarati, A. Ciampichetti, and S. Jajodia. Information flow control in object-oriented systems. *IEEE Transactions on Knowledge and Data Engineering, 09 (04) : 524-538, 1997.*

[**BER98**] E. Bertino, S. C. Vimercati, E. Ferrari and P. Samarati. Exception-Based Information Flow Control in object-oriented systems. *ACM Transaction on information and system security, vol.1, N°.1, pages 26-65. Nov 1998.*

[**BER02**] Berinato, S. "Debunking the Threat to Water Utilities," *CIO Magazine, March 15, 2002.*

[**BER06**] N. Berrehouma. Flexible information flow control for web services. *Master thesis, Bejaia University, October 2006.*

[**BLI02**] B. Li. Analysing information–flow in java Program based on slicing technique. *ACM SIGSOFT Software Engineering Notes, 27 (5), 98-103, September 2002.*

[**BOE85**] W. E. Boebert and C. T. Ferguson. A partial solution to the discretionary Trojan horse problem. *In proceedings of Eighth Nat'l Computer Security Conf., pages 141-144, Gaithersburg, Md, 1985.*

[**BRA84**] G. Bracha. An asynchronous $\lfloor (n-1)/3 \rfloor$-resilient consensus protocol. *In Proceedings of the 3rd ACM Symposium on Principles of Distributed Computing - PODC'84, pages 154–162, Aug. 1984.*

[**BRY94**] Bryce (C), Etude et mise en œuvre des propriétés de sécurité, *Rennes University, France, 1994.*

[**BRY95**] C. Bryce, J.P. Banâtre, D. Le Métayer: An Approach to Information Security in Distributed Systems. *IRISA/INRIA-Rennes University, France, page 384-394. 1995.*

[**BYR00**] Byres, E. "Designing Secure Networks for Process Control," *IEEE Industry Applications Magazine, vol. 6, no. 5, September/October, 2000.*

[**BYR02**] Byres, E., and D. Hoffman. "IT Security and the Plant Floor," *InTech Magazine, December 2002.*

[**BYR04**] E. Byres and J. Lowe. The myths and facts behind cyber security risks for industrial control systems. *In Proc. of VDE Kongress, 2004.*

[**CAS02**] M. Castro and B. Liskov. Practical Byzantine fault-tolerance and proactive recovery. *ACM TOCS, 20(4):398–461, 2002.*

[**CER01**] Civil Engineering Research Foundation. "Protecting Infrastructure." In Designing and Managing Vulnerability. *Washington, DC: American Society of Civil Engineering, October 2001.*

[**CHÉ92**] M. Chérèc, D. Powell, P. Reynier, J.-L. Richier, and J. Voiron. Active replication in Delta-4. *In Proc. 22th Symp. on Fault-Tolerant Computing - FTCS'92, pages 28–37, July 1992.*

[**CIP01**] Committee on Critical Infrastructure Protection. Securing Oil and Natural Gas Infrastructures in the New Economy. *National Petroleum Council, June, 2001.*

[**CUP93**] F.Cuppens. A Logical Analysis of Authorized and Prohibited Information Flows. *In IEEE Symposium on Security and Privacy, Oakland, 1993*.

[**DAC03**] Dacey, R. Critical Infrastructure Protection: Challenges in Securing Control Systems *(GAO Document GAO-04-140T). October 1, 2003.*

[**DEN76**] Denning (D.), A Lattice Model of Secure Information Flow, *in Communications of the ACM, 19 (5), May 1976, pages 236-243.*

[**DES92**] Y. Desmedt and Y. Frankel. Shared generation of authenticators and signatures (extended abstract). *In Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology - CRYPTO'92, pages 457–469, Aug. 1992.*

[**DHS03**] Department of Homeland Security. Procedures for Handling Critical Infrastructure Information; *Proposed Rule (Federal Register, 6 CFR Part 29). Washington, DC: Department of Homeland Security, April 15, 2003.*

[**DOB86**] Dobson, J., Randell, B.: Building reliable secure computing systems out of unreliable insecure components. *In: Proceedings of the International Symposium on Security and Privacy, IEEE (1986) 187–193.*

[**DOU99**] A. Doudou, B. Garbinato, R. Guerraoui, and A. Schiper. Muteness failure detectors: Specification and implementation. *In Proc. of the 3rd European Dependable Computing Conference, pages 71–87, September 1999.*

[**DZU05**] D. Dzung, M. Naedele, T. P. V. Hoff, and M. Crevatin. Security for industrial communication systems. *Proc. of the IEEE, 93(6):1152–1177, 2005.*

[**EET06**] M. van Eeten, E. Roe, P. Schulman, and M. de Bruijne. The enemy within: System complexity and organizational surprises. *In M. Dunn and V. Mauer, editors, Int. CIIP Handbook 2006, volume II, pages 89–110. CSS, ETH Zurich, 2006.*

[**FRA85**] Fraga, J.S., Powell, D.: A fault- and intrusion-tolerant file system. *In: proceedings of the 3rd International Conference on Computer Security. (1985) 203–218.*

[**GAR92**] T. D. Garvey and T. F. Lunt. Cover Stories for Database Security. *In S. Jajodia and C. Landwehr, editors, Database Security, 5: Status and Prospects. North-Holland, 1992. Results of the IFIP WG 11.3 Workshop on Database Security.*

[**GRA72**] G. S. Graham and P. J. Denning. Protection- principles and practice. *AFIPS: Conference Proceedings Volume 40 Spring Joint Computer Conference, 40: 417-429, 1972.*

[**GRA91**] J. W. Gray III. Toward a mathematical foundation for information flow security. *Sp, 00: 21, 1991.*

[**HOA69**] Hoar (C.A.R.), "An Axiomatic Basis for Computer Programming", *in communications of the ACM, volume 12 (10) ,October 1969, pages 576-583.*

[**HOA78**] Hoar (C.A.R.), "Communicating Sequential Processes", *in communications of the ACM, volume 21 (8), August 1978, pages 666-674.*

[**I3E94**] IEEE *''Institute of Electrical and Electronics Engineer*s'' Standard Definition, Specification, and Analysis of Systems Used for Supervisory Control, Data Acquisition, and Automatic Control (IEEE StandardC37.1-1994).

[**KEN05**] S. Kent. IP Authentication Header. *RFC 4302 (Proposed Standard), Dec. 2005.*

[**KOT07**] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong. Zyzzyva: Speculative Byzantine fault tolerance. *In Proc. of the 21st ACM Symp. On Operating Systems Principles (SOSP'07), Oct. 2007.*

[**KRZ79**] Krzysztof R. Apt, Nissim Francez, Willem P. de Roever. A Proof System For Communicating Sequential Processes. *Technical report, departments of computer science, Technion, Haifa, Palestine, Rotterdam and Utrecht Universities, Netherlands. August 1979.*

[**LHF78**] L. H. Fink and K. Carlsen. Operating under stress and strain. *IEEE Spectrum, Mar. 1978.*

[**LIN99**] T. Lindholm and F. Yellin. The java virtual machine specifications. *Sun Microsys-tems Inc., 1999.*

[**MAD05**] V. Madani and D. Novosel. Getting a grip on the grid. *IEEE Spectrum, 42(12):42–47, Dec. 2005.*

[**MCC90**] C.J. McCollumn, J.R. Messing and L. Notargiacomo. Beyond the pale of mac and dac – defining new forms of access control. *In proc. of the 1990 IEEE Computer Society Symposium on Security and Privacy, Oakland, CA.*

[**MCL90**] J. McLean. Security models and information flow. *In IEEE Symposium in Security and Privacy, pages 180-189, 1990.*

[**MON06**] H. Moniz, N. F. Neves, M. Correia, and P. Verissimo. Randomized intrusion-tolerant asynchronous services. *In Proc. of the Int. Conf. on Dependable Systems and Networks (DSN'06), pages 568–577, jun 2006.*

[**MYE00**] A.C. Myers, B. Liskov. Protecting privacy using the decentralised label model. *ACM Transactions on Software Engineering and Methodology, 9 (4), 410-442, October 2000.*

[**MYE97**] A.C. Myers, B. Liskov. A decentralised model for information flow control. *In Proc. Of the 16th ACM Symposium on Operating systems principles, Saint-Malo France, October 1997.*

[**MYE99**] A.C. Myers. Jflow: Practical mostly-static information flow control. *In Proc. Of the 26th ACM SIGPLAN-SIGACT Symposium on principles of Programming Languages, 1999.*

[**OBE06**] R. R. Obelheiro, A. N. Bessani, L. C. Lung, and M. Correia. How practical are intrusion-tolerant distributed systems? *DI-FCUL TR 06–15, Dep. Of Informatics, Univ. of Lisbon, Sept. 2006.*

[**OPP98**] R. Oppliger. Security at the internet layer. *IEEE Computer, 31(9):43–47, Sept. 1998.*

[**OSB02**] S.L. Osborn. Information flow analysis of an RBAC system. *In Proc of the 17th ACM Symposium on Access Control Models and Techniques, June 2002.*

[**POW92**] Powell, D.: Fault assumptions and assumption coverage. In Proceedings of the 22nd *IEEE International Symposium of Fault-Tolerant Computing*. 1992.

[**REI78**] Reitman (R.), Information Flow In Parallel Programs, *PhD Thesis, Cornell University, USA, 1978.*

[**SAN92**] R. Sandhu and S. Jajodia. Polyinstantiation for cover stories. In *European symposium on research in computer security, Toulouse, France, 1992. AFCET.*

[**SOU05**] P. Sousa, N. F. Neves, and P. Verissimo. How resilient are distributed f fault/intrusion-tolerant systems? *In Proc. Of Int. Conf. on Dependable Systems and Networks (DSN'05), pages 98–107, June 2005.*

[**SOU06**] P. Sousa, N. F. Neves, A. Lopes, and P. Verissimo. On the resilience of intrusion-tolerant distributed systems. *DI/FCUL TR 06–14, Dep. of Informatics, Univ. of Lisbon, Sept 2006.*

[**SOU07**] P. Sousa, A. N. Bessani, M. Correia, N. F. Neves, and P. Verissimo. Resilient intrusion tolerance through proactive and reactive recovery. *In Proc. of the 13th IEEE Pacific Rim Int. Symp. on Dependable Computing (PRDC'07), pages 373–380, Dec. 2007.*

[**STO81**] A. Stoughton. Access flow: A protection model which integrates access control and information flow. *In Proceeding of the 1981 IEEE Computer Society Symposium on Security and Privacy, pages 9-18, Oakland, CA, 1981.*

[**SUN07**] Sun Microsystems Inc. Java Security architecture. April 2007. Available at http://java.sun.com/j2se/1.3/docs/guide/security/spec/security-pec.doc12.html.

[**TAR06**] Z. Tari, P. Bertok, and D. Simic. A dynamic label checking approach for information flow control in Web Services. *International Journal for Web Services Research, 3 (1): 1-28, 2006.*

[**TUS96**] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *J. ACM, 43(2), March 1996.*

[**VER01**] Verissimo, P., Rodrigues, L.: Distributed Systems for System Architects. *Kluwer Academic Publishers (2001).*

[**VER03**] P. Verissimo, N. F. Neves, and M. P. Correia. Intrusion-tolerant architectures: Concepts and design. *In Architecting Dependable Systems, volume 2677 of LNCS. 2003.*

[**VER06**] P. Verissimo, N. F. Neves, and M. Correia. CRUTIAL: The blueprint of a reference critical information infrastructure architecture. *In Proc. of CRITIS'06 1st Int. Workshop on Critical Information Infrastructures Security, Aug. 2006.*

[**VER+06**] P. Verissimo. Travelling through wormholes: a new look at distributed systems models. *SIGACT News, 37(1), 2006.*

[**VOL96**] D. Volpano, G. Smith, and C. Irvine. A sound type system for secure flow analysis. *Journal of Computer Security, 4 (3): 167-187. December 1996.*

[**WAL00**] D. S. Wallach, A. W. Appel, and E. W. Felten. Safkasi: A security mechanism for language-based systems. *ACM Trans. Softw. Eng. Methodol., 9 (4): 341-378, 2000.*

[**WIL06**] C. Wilson. Terrorist capabilities for cyber-attack. *In M. Dunn and V. Mauer, editors, Int. CIIP Handbook 2006, volume II, pages 69–88. CSS, ETH Zurich, 2006.*

[**ZHO02**] L. Zhou, F. Schneider, and R. Van Rennesse. COCA: A secure distributed online certification authority. *ACM TOCS, 20(4):329–368, Nov. 2002.*

Résumé

La tolérance d'intrusion (TI) et le contrôle de flux d'information (CFI) sont des mécanismes de sécurité très importants pour les environnements d'infrastructures critiques et distribués tels que les systèmes SCADA. La TI comme corps de la connaissance est le catalyseur principal de l'évolution du secteur de la fiabilité. Nous croyons que la tolérance de fautes sera témoin d'une évolution extraordinaire, qui aura l'applicabilité dans tous les domaines et notamment lesquels reliés à la sécurité. Le CFI quant à lui, assure que les flux d'information au sein d'un système ne provoquent pas des divulgations ou des destructions d'information. Il a été introduit pour les langages de programmation et il est utilisé dans d'autres domaines comme les bases de données et les systèmes distribués.

Dans ce projet, nous avons amélioré un modèle de CFI basé sur l'analyse du code proposé par Bryce, Banâtre & Le Métayer et qui assure la confidentialité des informations sensibles relatives aux environnements critiques, nous avons défini des nouvelles règles et leurs sémantiques de flux. En outre, nous avons intégré un modèle de TI pour la protection des systèmes SCADAs (fonctionne comme un pare-feu) et un autre de CFI afin de renforcer la résilience de tels systèmes (i.e. énergie, eau, gaz, etc.) contre les attaques malveillantes.

Mots clé : Les Systèmes SCADAs, Sécurité, Pare-feux, Attaques, Tolérance d'Intrusion, Contrôle de Flux d'Information.

Abstract

Intrusion tolerance (IT) and information flow control (IFC) are very important mechanisms of safety for distributed and critical infrastructures environments such as SCADA systems. IT as body of knowledge is the principal catalyst of the evolution of reliability sector. We believe that the faults tolerance will be pilot of an extraordinary evolution, which will have applicability in all fields and in particular those related to security. The IFC as for him ensures that information flows within a system do not cause disclosures or destruction of information. It was introduced for the programming languages and it is used in other fields as distributed and data bases systems.

In this project, we improved a model of IFC based on code analysis suggested by Bryce, Banâtre & Le Métayer and that ensures confidentiality of sensitive information relating to critical environments, we defined new rules and their flow semantics. Moreover, we integrated an IT model for SCADAs systems protection (like a firewall) and another of IFC in order to enhance resilience of such systems (i.e. energy, water, gas, etc) against malicious attacks.

Key words: SCADA Systems, Security, Firewalls, Attacks, Intrusion Tolerance, Information Flow Control.