Specification-based Intrusion Detection for Hierarchical Hybrid Industrial Control Systems

Estelle Hotellier

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

École doctorale : MSTII - Mathématiques, Sciences et technologies de l'information, Informatique
Spécialité : Informatique
Unité de recherche : Centre de recherche Inria de l'Université Grenoble Alpes

Détection d'Intrusions basée sur les spécifications pour les Systèmes de Contrôle-Commande Industriels Hiérarchiques Hybrides

Specification-based Intrusion Detection for Hierarchical Hybrid Industrial Control Systems

Présentée par :

Estelle HOTELLIER

Direction de thèse :

Stéphane MOCANU MAITRE DE CONFERENCES HDR, GRENOBLE INP	Directeur de thèse
Franck SICARD INGENIEUR DE RECHERCHE, NAVAL GROUP	Co-encadrant de thèse
Julien FRANCQ INGENIEUR DE RECHERCHE, NAVAL GROUP	Co-encadrant de thèse

Rapporteurs :

HERVE DEBAR
PROFESSEUR, TELECOM SUDPARIS
VALERIE VIET TRIEM TONG
PROFESSEURE, CENTRALESUPELEC RENNES

Thèse soutenue publiquement le **9 avril 2024**, devant le jury composé de :

MARIE-LAURE POTET, PROFESSEURE DES UNIVERSITES, GRENOBLE INP	Présidente
STEPHANE MOCANU, MAITRE DE CONFERENCES HDR, GRENOBLE INP	Directeur de thèse
HERVE DEBAR, PROFESSEUR, TELECOM SUDPARIS	Rapporteur
VALERIE VIET TRIEM TONG, PROFESSEURE, CENTRALESUPELEC RENNES	Rapporteure
ISABELLE CHRISMENT, PROFESSEURE, TELECOM NANCY	Examinatrice
LUDOVIC ME, SENIOR SCIENTIST, CENTRE INRIA DE L'UNIVERSITE DE RENNES	Examinateur

Invités :

FRANCK SICARD
INGENIEUR DE RECHERCHE, NAVAL GROUP
JULIEN FRANCQ
INGENIEUR DE RECHERCHE, NAVAL GROUP

Remerciements

Je tiens à remercier toutes les personnes qui ont contribué à la réalisation de cette thèse. Le chemin vers l'achèvement de ce doctorat fut long (parfois douloureux), mais le soutien que j'ai reçu a rendu cette aventure enrichissante et mémorable.

Je tiens tout d'abord à exprimer ma reconnaissance envers mon directeur de thèse, Stéphane Mocanu, ainsi qu'à mes encadrants, Franck Sicard et Julien Francq. Leur soutien constant et leur engagement ont été des piliers fondamentaux tout au long de ce processus de recherche. Merci pour votre accompagnement scientifique et humain, j'ai beaucoup appris à vos côtés pendant ces trois années.

Je suis également reconnaissante envers les membres de mon jury d'avoir accepté d'évaluer mes travaux. Merci à Hervé Debar et Valérie Viêt Triêm Tông, d'avoir endossé le rôle de rapporteurs. Merci également à Isabelle Chrisment, Marie-Laure Potet et Ludovic Mé. Merci à tous pour vos retours constructifs.

Merci à mon équipe de recherche à Naval Group de m'avoir accordé votre confiance et de m'avoir offert les moyens de réaliser ces travaux de recherche pendant trois ans au Naval Cyber Laboratory (NCL). Sans vos discussions animées, vos blagues douteuses et vos confcall dans l'openspace, les heures passées devant mon ordinateur auraient été bien plus longues. Mention particulière à mon stagiaire Nahi Boukhobza pour son aide scientifique précieuse. Mes remerciements vont également à mes collègues de l'inria et à tous ceux qui ont partagé leurs connaissances et leur expérience avec moi.

Un énorme merci à ma famille et mes amis, pour avoir supporté mes humeurs changeantes et mes conversations incessantes sur des sujets ésotériques. Votre soutien inconditionnel m'a été très précieux. Mention spéciale à Amandine Ferret et Alban Cotton. Merci à vous et vos familles de m'avoir accueillie à Grenoble.

À Pierre, merci pour ta présence durant cette aventure, avec la certitude de nombreuses autres à venir.

Abstract

This thesis tackles the question of cybersecurity through network intrusion detection for Industrial Control Systems (ICSs). Our interest arises from the increasing number of cybersecurity incidents targeting ICSs and the need to detect attacks that manipulate the physical process. Such attacks are called process aware attacks: they are sophisticated cyberattacks aiming at disrupting the physical process and inducing incorrect behaviors of the system.

In this manuscript, we propose a significant contribution by developing a specification-based, process aware, Intrusion Detection System (IDS) for ICSs. ICSs are distributed and hierarchical control systems, built on top of local control loops which are the system's elementary building blocks.

Our approach aims to link safety specifications and security properties. Thus, we use international and industry standards specifications concerning local safety, global safety and networks of the industrial process, in order to systematically obtain security properties. The obtained security properties are cybersecurity related requirements. They are translated into security patterns in order to be runtime monitored by our network IDS. We rely on specification language formalism such as Linear Temporal Logic (LTL), Metric Temporal Logic (MTL) and Signal Temporal Logic (STL) to express temporal properties and tackle the hybrid dynamics of ICSs. Our approach relies on a monitoring framework, capturing network traffic between the local loops and the distributed control level (i.e. on fieldbuses notably), as well as between distributed control and supervisory control.

In our work, we also explore a distributed deployment framework for our IDS. This is motivated by the necessity to deploy multiple IDS instances when more detection capacities are required, in order to monitor larger industrial systems in spread out environments. Our distributed framework consists in structurally identical IDS instances deployed across the system and coordinated in a way that they can monitor system's local and global security properties.

We implemented and evaluated our IDSs on physical ICS testbeds. We experimentally show that our IDS detects a large spectrum of attacks. We showed the detection capacities of our distributed deployment. We show that our approach is scalable since its detection response time as a function of the number of monitored security patterns, is linear. Finally, we illustrated our approach's extensibility by deploying it on two distinct ICS testbeds.

Keywords: Network intrusion detection, industrial control system, cybersecurity, anomaly detection, behavioral detection, runtime verification, distributed detection.

Résumé

Cette thèse apporte une contribution au domaine de la cybersécurité, en particulier la détection d'intrusions réseau pour les systèmes de contrôle-commande industriels (ICSs en anglais). Notre intérêt découle de l'augmentation des incidents de cybersécurité ciblant les ICS et de la nécessité d'identifier les attaques qui manipulent les processus physiques de ces systèmes (*process aware* en anglais). Ces attaques sont des cyberattaques sophistiquées conçues pour cibler les processus physiques et provoquer des comportements incorrects du système.

Dans ce manuscrit, nous développons un système de détection d'intrusions (IDS) adapté aux ICSs. Notre approche est basée sur des spécifications et prend en compte la connaissance des processus physiques des systèmes. Les ICSs sont des systèmes de contrôle distribués et hiérarchiques construits à partir de boucles de contrôle locales, qui constituent les blocs élémentaires de ces systèmes.

Notre méthodologie vise à établir un lien entre les spécifications de sécurité et les propriétés de sûreté d'un système. Pour cela, nous utilisons des spécifications issues de normes internationales et de standards industriels concernant la sécurité locale et globale, ainsi que les spécifications des protocoles réseau, dans les processus industriels. L'extraction de ces spécifications est réalisée de manière systématique et permet l'obtention de propriétés de sûreté (*security properties* en anglais). Ces propriétés de sûreté forment des exigences ayant une signification au niveau du réseau et peuvent être évaluées pendant l'exécution du système. Nous utilisons des formalismes tels que la logique temporelle linéaire (LTL), la logique temporelle métrique (MTL) et la logique temporelle pour les signaux (STL) permettant de formaliser des propriétés temporelles et de prendre en compte les dynamiques hybrides propres aux ICSs. Notre approche est réalisée à l'aide de captures de trafic réseau entre les boucles locales et le niveau de contrôle distribué (sur les bus de terrain notamment), ainsi qu'entre le contrôle distribué et la supervision.

Dans un second temps, nos travaux explorent une configuration distribuée pour le déploiement de notre IDS. Cette architecture est motivée par la nécessité de déployer plusieurs instances d'IDS lorsque des capacités de détection accrues sont requises, par exemple pour une détection sur des systèmes industriels de plus grande dimension. Notre approche distribuée se compose de plusieurs instances de notre IDS, structurellement identiques, déployées dans l'ensemble du système et coordonnées pour surveiller des propriétés de sûreté locales et globales du système.

Nous avons mis en œuvre et évalué notre approche sur des plateformes expérimentales de systèmes industriels, démontrant expérimentalement sa capacité à détecter un large spectre d'attaques. Nous avons présenté les capacités de détection de notre déploiement distribué. Nous avons montré la scalabilité de notre approche, puisque la relation entre le temps de détection et le nombre de propriétés de sûreté évaluées est linéaire. Enfin, nous avons illustré son adaptabilité lorsque notre approche est déployée sur différents systèmes.

Mots-clés: Détection d'intrusions réseau, système de contrôle-commande industriel, cybersécurité, détection d'anomalies, détection comportementale, vérification à l'exécution, détection distribuée.

Résumé Général

Le présent manuscrit étant rédigé en anglais, voici un résumé des contenus de chacun des chapitres, en français.

PARTIE I

Chapitre 1. Systèmes de contrôle-commande industriels (ICS)

Ce premier chapitre introduit la notion de système de contrôle-commande industriel (ICS). Tout d'abord, une définition et les éléments nécessaires au contexte sont présentés. L'architecture générale d'un ICS est détaillée et les différences entre les technologies opérationnelles (OT) – constituant principalement les ICSs – et les technologies de l'information (IT) sont discutées. Nous donnons les motivations qui nous ont poussés à réaliser ces travaux et soulignons la nécessité de sécuriser les ICSs. Ensuite, quelques exemples d'attaques récentes sont détaillés avec une mise en évidence des conséquences désastreuses que ces attaques peuvent avoir sur les processus physiques. Enfin, nous discutons des aspects de la sécurité dans les ICSs, en définissant les concepts et en donnant des mesures de sécurité générales.

Chapitre 2. Etat de l'art

Ce chapitre commence par définir et caractériser le concept de système de détection d'intrusion (IDS) en mettant l'accent sur les IDSs basés sur les anomalies puisque c'est le type d'IDS que nous avons développée dans notre approche.

Puisque nos travaux utilisent des principes de la *vérification à l'exécution* des systèmes (*runtime verification* ou *runtime monitoring* en anglais), nous présentons l'état de l'art de ce domaine pour permettre une meilleure compréhension de nos contributions.

Enfin, le thème de la détection d'intrusion distribuée est étudié car il concerne une de nos contributions.

Tout au long du chapitre, les travaux existants dans la littérature sont présentés et discutés. Finalement, les lacunes et points encore inexplorés de la littérature sont identifiés. Cela nous permet de positionner nos travaux et de répondre à ces verrous scientifiques et techniques.

PARTIE II

Chapitre 3. Plateformes expérimentales d'ICSs

La recherche en matière de cybersécurité permet d'anticiper les attaques futures. La validation de nouvelles mesures de sécurité doit être menée sur des plateformes expérimentales réalistes afin d'obtenir un retour d'information précis et fructueux pour la cyberdéfense. C'est la raison pour laquelle ce chapitre est dédié aux plateformes expérimentales (*testbeds* en anglais) représentant des systèmes de contrôle-commande industriels. Dans ce chapitre, deux plateformes expérimentales de systèmes industriels sont présentées en détail puisqu'elles sont utilisées pour l'application et l'évaluation de nos contributions.

Chapitre 4. Approche de détection

Ce chapitre apporte des contributions aux principales faiblesses identifiées dans la littérature concernant les systèmes de détection des intrusions pour les ICSs. Il présente notre approche de détection des intrusions.

Par conséquent, le but de ce chapitre est d'exposer la méthodologie à suivre pour obtenir une approche de détection d'intrusions comportementale qui : (i) prend en compte la connaissance du processus physique et peut s'appliquer au niveau du bus de terrain (tirant ainsi parti de la surveillance de l'état des actionneurs/capteurs et de certains états/contextes des composants) ; (ii) présente une construction systématique du modèle de détection, basée sur les spécifications présentes dans les normes internationales et standards industriels (réduisant ainsi le coût de la construction du modèle de détection) ; (iii) repose sur des formalismes de logique temporelle qui couvrent la dynamique hybride des ICSs et (iv) présente une large plage d'observation grâce à un déploiement distribué de détection dans différentes boucles locales du système, couvrant ainsi un plus grand nombre d'attaques.

Ce chapitre présente également le modèle de la menace. Il donne ensuite la vue d'ensemble de notre approche avant de détailler sa méthodologie de déploiement.

Chapitre 5. Détection sur un système industriel hiérarchique simple

Ce chapitre vise à structurer l'approche sur un système industriel hiérarchique simple. Le cas d'usage sur lequel nous nous appuyons est un robot cartésien (qui est un système hiérarchique simple). Il fait partie d'une des plateformes expérimentales décrites dans le Chapitre 3.

Ainsi, nous détaillons étape par étape la méthodologie pour déployer notre système de détection. Nous fournissons des exemples concrets pour le déploiement des propriétés de sécurité du modèle de détection. Par ailleurs, nous donnons des détails sur la mise en œuvre de la vérification des propriétés de sécurité pendant l'exécution du système. Ensuite, nous présentons le déploiement logiciel de notre architecture. Enfin, nous évaluons notre approche et discutons les résultats obtenus.

Chapitre 6. Détection distribuée

Dans le chapitre précédent, nous avons présenté notre IDS réseau pour des systèmes hiérarchiques simples. Nous avons évalué notre approche pour une unique instance du système de détection. Toutefois, si des capacités de détection plus importantes sont nécessaires (pour un cas d'utilisation plus large par exemple), l'approche peut être distribuée sur plusieurs

machines afin d'équilibrer la charge de détection des intrusions. Cependant, cela conduit inévitablement à des vues restreintes du système (c'est-à-dire des vues locales). Le défi consiste à gérer ces vues locales afin d'être capable de vérifier des propriétés de sécurité globales du système ; c'est le sujet du sixième chapitre.

Comme dans le chapitre précédent, des exemples concrets sont fournis pour le déploiement distribué de notre méthodologie de détection en s'appuyant notamment sur une plateforme expérimentale de plus grande échelle. Nous décrivons le déploiement logiciel de notre architecture. Enfin, nous évaluons notre approche et discutons les résultats obtenus.

Contents

List of Figures

List of Tables

Introduction

This thesis concerns intrusion detection in Industrial Control Systems (ICSs). An ICS consists of combinations of control components (e.g., electrical, mechanical, hydraulic, pneumatic) acting together to control physical processes in order to achieve an industrial objective. ICSs form the backbone of critical infrastructures whose role extends across vital sectors including energy production and distribution, water treatment and supply, transportation, and diverse manufacturing applications. Sometimes, ICSs are designed as OT (Operational Technology) – as they control physical system, in contrast with classical IT (Information Technology) systems.

The evolution of ICSs unfolds against the backdrop of an increasingly integration with classical IT systems. If ICSs used to historically operate as isolated systems, it is no longer the case and they are now interconnected with Internet and have adopted standardized protocols, TCP/IP networking, common operating systems, etc. This convergence between OT and IT has propelled new services and capabilities in terms of local and global control of physical processes, but it had also exposed ICSs to unprecedented cybersecurity risks. Therefore, this interconnection technology introduces its own failure vectors and vulnerabilities. Such vulnerabilities, if exploited, can result in substantial damage impacting humans, systems and environment.

A defining characteristic setting ICS apart from IT systems lies in the presence of a physical process. Hence, in an ICS, the manipulated data has a physical meaning. In recent years, this observation has led to the development of a novel category of attacks which specifically targets the physical process. We refer to them as *process aware* attacks. They require a deep knowledge of the physical process and they aim at provoking incorrect behaviors of physical processes. Among examples of such attacks, it is worth citing the well-known Stuxnet worm directed against Iranian Natanz Nuclear facility in 2010. This attack was the first one with high media coverage and it emphasized, for the first time, the need for cybersecurity within ICSs. Industroyer attack in 2016 has successfully affected the Ukrainian electric power grid for a few hours, depriving over 230,000 people of electricity. In 2021, Oldsmar water treatment facility in Florida was victim of a poisoning attempt in the city water supply. A variant of Industroyer attack, called INDUSTROYER.V2 was launched in 2022, with less success, probably due to improved Ukrainian cyberdefense practices since 2016.

The rise of process aware attacks on ICSs has underscored the necessity for security controls tailored to the unique characteristics of these systems. Due to very strict constraints on the response time needed by control functions and scarce computing resources, classical security controls like anti-viruses or firewalls are difficult to deploy on ICSs devices. Also, ICSs network communications are submitted to bounded transmission delays in order to ensure the timely realisation of control functions. In all cases, security controls in ICSs must not disrupt the normal functioning of the systems by introducing time latencies. In order to address these challenges and secure ICSs against process aware attacks, we identify in this manuscript *Intrusion Detection Systems* (IDSs) as a promising solution in the context of in-depth cyberdefense strategy. IDSs divide in two major categories: misuse-based and anomaly-based. The first category relies on specific recognition of malicious actions, whereas the second consists in building a reference model of the usual behavior of the system and detecting deviations from it. To go a step further, we identified in our work behavior-based

approaches to be very effective for intrusion detection in industrial systems and we adopt a *specification-based* approach. This consists in using a set of specifications and constraints describing the correct operation of the system being monitored.

Several ICS-specific behavior-based IDS approaches are explored within the research community. However, the state of the art of intrusion detection for ICSs reveals significant gaps. Approaches that attempt to detect process aware attacks by relying solely on the communication aspect of ICS networks with no comprehension of the physical process, have few conclusive detection results. On the contrary, some approaches encompass significant knowledge about the physical process but generally suffer from a manual characterization process for the creation of the detection model (this is particularly true for specification-based approaches). Moreover, many approaches rely on a limited model expressiveness and do not fully comprehend both the operational contexts and dynamics of the underlying industrial processes. Finally, few research effort tackles distributed deployments of the intrusion detection task, which results in poor scalability to spread out industrial environments. On the practical aspect, many approaches do not implement data collection at the closest to the physical process: data collection is often executed on TCP/IP networks that are easier to instrument than fieldbuses for instance. This results in a lower fidelity in the monitored data, since the latter may have been propagated across different communication medium. This thesis aims at addressing these gaps.

Contributions.

In this work, we present the following contributions:

- We propose a specification-based intrusion detection approach for hierarchical industrial control systems with hybrid dynamics. The main paradigm of our approach is to link safety specifications and security properties. Our approach aims at detecting security properties violations specific to OT, which lead to safety specification violations. The specifications are extracted from international and industry standards, in a systematical manner. Then, they are synthesized as monitors that evaluate execution traces of the system and raise alerts when a security property is violated.
- We extend the detection capabilities of Ethernet-based IDSs to fieldbus monitoring, namely Controller Area Network − CAN, and Modbus Remote Terminal Unit − RTU.
- We present a distributed deployment for our intrusion detection approach with the evaluation of local security properties and global security properties of the system. The distributed framework is constituted of several instances of our specification-based IDS.
- We conduct experimentation on two ICS testbeds, and provide extensive evaluation of our detection framework in terms of detection response time and detection capabilities. We evaluate the scalability of our solution by discussing the number of monitors and we provide details regarding extensibility.

Manuscript outline

The remainder of this manuscript is organized in two Parts. The first Part, divided in two chapters, sets the background and state of the art. The second Part presents our contributions in four Chapters.

Chapter 1 defines the main concepts, and the background necessary to understand and position our contributions. The characteristics of ICSs are provided, we give examples of major security incidents and the concept of securing ICSs is discussed. In Chapter 2, we

first provide a state of the art of intrusion detection systems for ICSs and discuss current limitations observed in the literature. Secondly, we introduce work on runtime verification that is a lightweight verification methodology we rely on for our contributions. Finally, we discuss distributed detection systems for ICSs.

The second Part is dedicated to our contributions. Chapter 3 presents the advantages of working on a physical testbed when investigating industrial system intrusion detection. This chapter presents the two testbeds used for the experimental application of our work. Chapter 4 details our detection framework. We first present our threat model, then we give the motivations for our framework and detail the methodology to deploy our approach. In Chapter 5, we provide step by step implementation of our IDS approach on a physical ICS testbed (presented in Chapter 3). This allows to evaluate our methodology, present the results and discuss them. In Chapter 6, we present the distributed deployment for our intrusion detection approach, we evaluate it on a physical ICS testbed and discuss the results.

Part I

BACKGROUND AND STATE OF THE ART

Chapter 1

Introduction to Industrial Control Systems

Contents

Introduction

This first Chapter introduces the important notion of Industrial Control System (ICS) which is the subject of this work. First, a definition and necessary elements of context are provided. The general architecture of ICSs is detailed and the differences between Operational Technology (OT) — mostly constituting ICSs — and Information Technology (IT) is discussed. Next, we provide the motivation of our work and we highlight the need for cybersecurity in ICS. Important definitions are given and examples of famous recent attacks are provided. Finally, we discuss security aspects in ICSs, defining concepts, and giving general security measures.

1.1 Industrial Control Systems

1.1.1 Terminology

An ICS consists of combinations of control components (e.g., electrical, mechanical, hydraulic, pneumatic) acting together to achieve an industrial objective [147] (e.g., manufacturing, transportation of matter, energy). This definition given by the National Institute of Standards and Technology (NIST) highlights the importance of the physical aspect of such

systems. The aim of an ICS is to produce an output, generally referred to as the *process* (or *physical process*). The French Network and Security Agency, Agence Nationale de la sécurité des systèmes d'information (ANSSI), gives a similar definition and adds the notion of human interaction with such systems. The ANSSI state that an ICS is the *set of human and material resources designed to control or operate a group of sensors and actuators* [38]. ICSs cover a wide range of activity sectors such as electrical, water and wastewater, oil and natural gas, chemical, transportation, pharmaceutical, pulp and paper, food and beverage, and discrete manufacturing (e.g., automotive, aerospace, durable goods) industries [147].

Moreover, ICSs are sometimes designed as OT in contrast with classical IT systems. As a matter of fact, NIST even revised the title of its reference technical report from *Guide to Industrial Control Systems (ICS) Security* [146] in 2015 to *Guide to Operational Technology (OT) Security* [147] in 2022. The authors define OT as *a broad range of programmable systems and devices that interact with the physical environment (or manage devices that interact with the physical environment). These systems and devices detect or cause a direct change through monitoring and/or control of devices, processes, and events. Examples include industrial control systems, building automation systems, transportation systems, physical access control systems, physical environment monitoring systems, and physical environment measurement systems.*

There is a common ground in these definitions, however the notion of ICS remains a broad concept and several variants exist such as Cyber Physical Systems (CPSs). A CPS [46, 105, 101] consists of a physical system and a cyber system. It results from an integration of physical processing, sensing, computation, communication and control. According to NIST definition, a CPS regroups *interacting digital, analog, physical, and human components engineered for function through integrated physics and logic.* According to this definition, an ICS is nothing less than a CPS applied to the industrial world.

Finally, alternative denominations for ICS can be observed in the literature such as Supervisory Control And Data Acquisition (SCADA) or Distributed Control System (DCS). According to the NIST [147], SCADA and DCS are different types of control systems. A DCS *refers to control achieved by intelligence that is distributed around the process to be controlled, rather than by a centrally located single unit.* A SCADA designates *a computerized system that is capable of gathering and processing data and applying operational controls over long distances.* Although quite close in their definitions, the difference between these two notions lies in their geographical distribution. A DCS is used to control production systems within the same geographic location for industries, whereas a SCADA system supports multiple remote stations and control centers based in several geographic locations [147]. Furthermore, it is worth mentioning the fact that SCADA is an ambiguous notion since it can either designate a control center, the software used for the supervisory control or the overall industrial control/command system. The various uses depends on the culture and the context on which they are employed.

However, the limits between aforementioned denominations are not always straightforward, especially as the technologies deployed in all these systems are increasingly converging. In this manuscript, we will only use the term ICS and introduce general characteristics common to every variant.

1.1.2 Architecture and Characteristic

The typical architecture of an industrial information system was synthesized for the first time around the 80s with a Reference Model for Computer Integrated Manufacturing (CIM) [155]. This major standardization effort was created during the International Purdue Workshop on Industrial Computer Systems, organized by The Purdue Laboratory for Applied Industrial Control of Purdue University, West Lafayette, Indiana. Its full name, according to its authors is *The Purdue Reference Model for CIM* but it is commonly known as *Purdue Model*. This model is used in ISA95 and ISA99 series of standards, and then recalled in

IEC 62264 [88]. Purdue Model is hierarchical and shows distributed control functions. It is structured in 6 levels: (i) Level 5: Corporate Strategy (strategy, direction), (ii) Level 4: Operational Management (planning), (iii) Level 3: Process Management (scheduling, inter-unit management), (iv) Level 2: Supervisory Control (unit management), (v) Level 1: Basic Control and (vi) Level 0: Operative Part. The reference architecture for enterprise is illustrated in Figure 1.

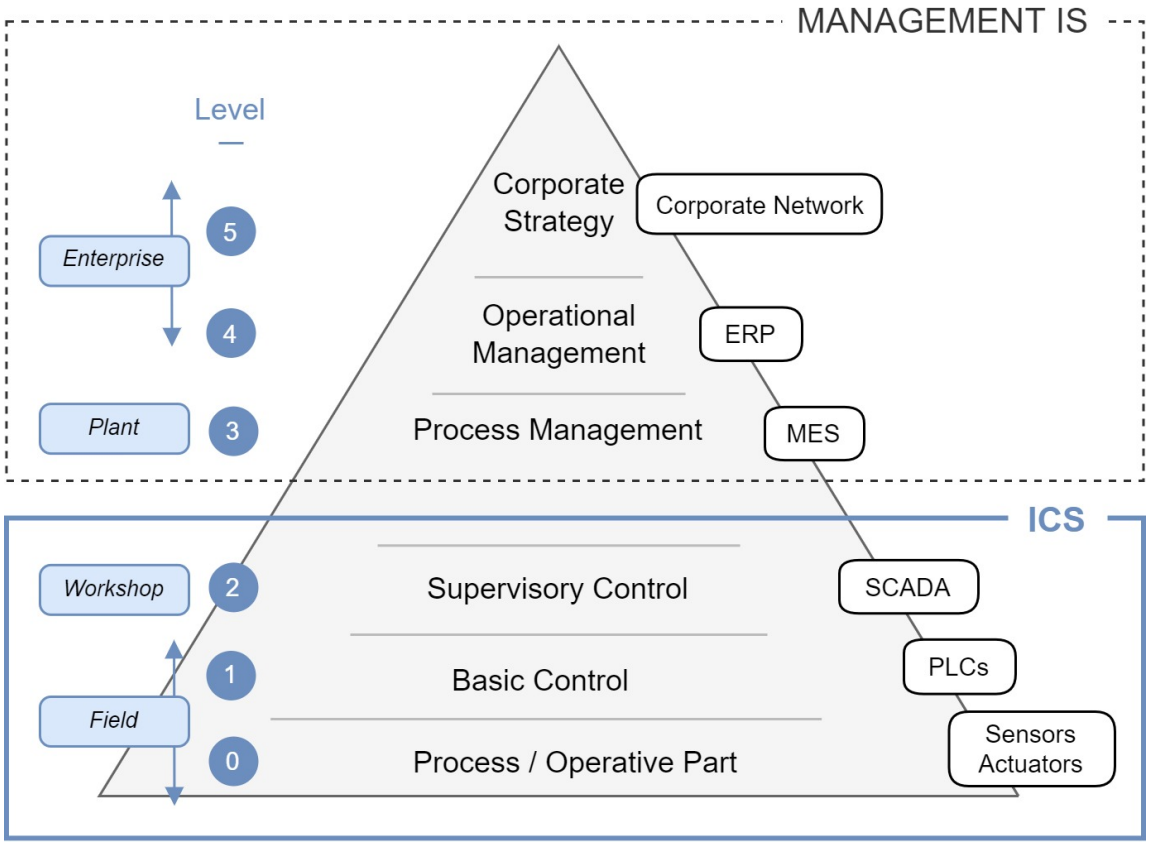

Figure 1: Illustration of Purdue Model

Each level exchanges information with its upper and lower levels. The higher the level, the greater the volume of data to be processed. As a consequence, the reaction time increases along with the understanding of the context. Each level is composed of characteristic elements that are indicated on the right side of Figure 1. On the higher levels (Levels 3 to 5), the Corporate Strategy relies on a Corporate Network, the Operational Management uses Enterprise Resource Planning (ERP) tools and the Process Management can be implemented with Manufacturing Execution System (MES) or other process scheduling tools. These three levels constitute the Management Information System (MIS) of the organization. Concurrently, the lower levels of the architecture form the ICS. The Supervisory Control (Level 2) uses SCADA equipment. The Basic Control (Level 1) is made through Programmable Logic Controllers (PLCs). And the Process and Operative Part (Level 0) is composed of field devices such as sensors and actuators.

ICS architecture

While Purdue Model describes the full enterprise architecture, we mainly focus on the ICS in the present work which represents Levels 0 to 2 of Purdue Model. Following the whole model, an ICS shows a hierarchical and distributed control strategy. Indeed, control and monitoring parts are distributed both vertically and horizontally across the system. Such a complex process is composed of elementary sub-processes which can be locally controlled.

Figure 2 details the typical architecture of an ICS.

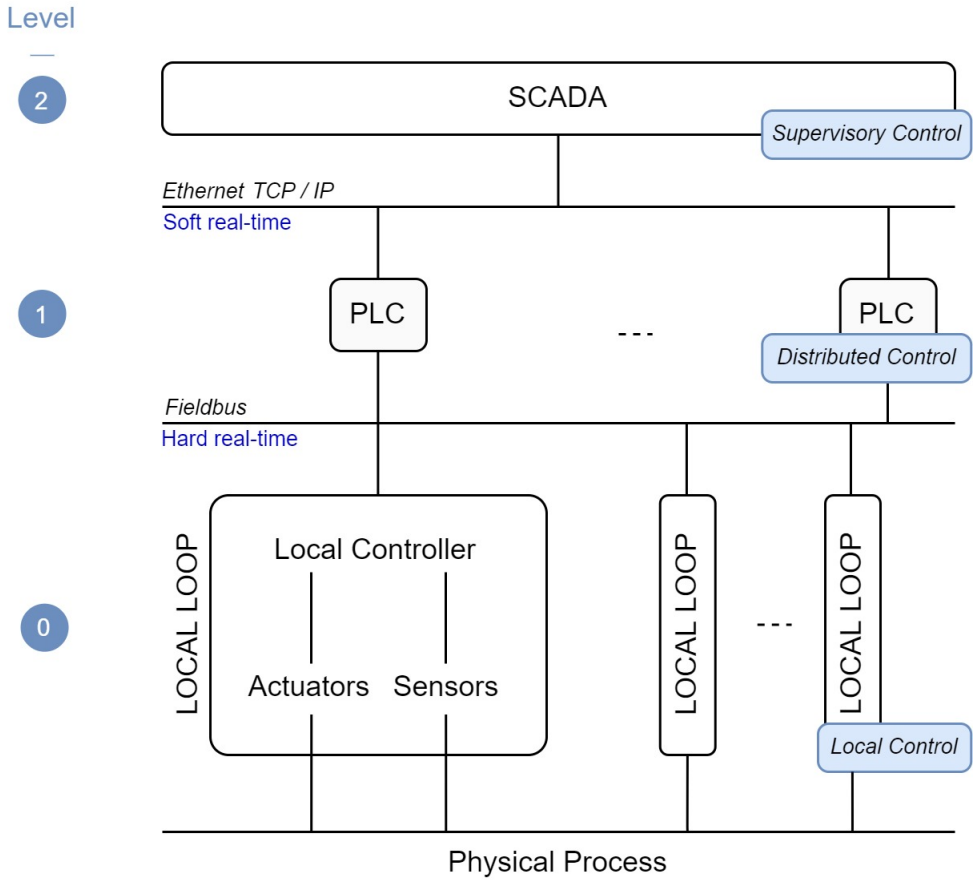

Figure 2: Typical architecture of a complex ICS

Level 0. The elementary building block of the ICS is the *local loop*, which is an autonomous sub-process together with its local controller. This local controller is directly interacting with the physical process via sensors and actuators. Typically, sensors and actuators are directly wired to the controller although some networked solutions may exist. The controller periodically acquires data from sensors and applies computed controls on actuators.

Level 1. Higher level controllers compute setpoints for the local controllers in order to achieve more global control objectives like synchronization of local loops. These higher level controllers are usually PLCs (Programmable Logic Controllers). More details on the functioning of PLCs are provided in Appendix A.

Level 2. At the highest level, the SCADA system is providing a global view of the system's state and activities. It is responsible of the monitoring and coordination between PLCs. The information from lower levels is presented to human operators in a visual way through Human Machine Interfaces (HMIs). An HMI displays graphical indicators or alarm notifications; it also allows the human operator to act on the physical process. Note that HMIs are used to provide a global view of the system at Level 2 but it can also give a local view of the local loop at Level 1.

To give an example of such a complex ICS, consider a two axis positioning system. Two local loops are in charge of controlling speed and position for the two motion axis, while a PLC is in charge of the trajectory control. An HMI allows the operator to change the target

position or trajectory.

Industrial communication protocols

At the different layers, the communication between devices requires different real-time performance levels. Actuators and sensors (Level 0) exchange data with their local controller, either through direct connections (using point to point or star topologies for example), or through broadcasting (using a bus topology for example). Examples of field-level communication protocols include AS-i bus[1], or Profibus DP[2]. At this level, the volume of exchanged data is low, and exchanges are mostly cyclic. The communication meets hard real-time performances and transfer time is usually of the order of milliseconds (ms). The communication between local loops (Level 0) and PLCs (Level 1) also meets hard real-time performance and uses specialized network solutions such as fieldbuses. Examples of fieldbuses are Modbus Remote Terminal Unit (RTU)[3] or Controller Area Network (CAN)[4]. At Level 1, the communication between PLCs for distributed control applications meets soft real-time requirements. Traffic is characterized, albeit in a lesser extent compared to Level 0, by its periodicity and low data volume. The transfer time is usually of the order of 100 ms. The communication between PLCs (Level 1) and supervisory control (Level 2) generally meets soft real-time performance as well, TCP/IP-based solutions can be used. At the higher level, SCADA communications (Level 2) begin to resemble classical IT systems communication. The volume of data exchanges is significant and time requirements are low. Consequently, no real-time is required and the transfer time is usually of the order of the second.

Hybrid dynamic of ICSs

A real-life industrial control system exposes "mixed" (also called *hybrid*) dynamics. Indeed, any system evolves through time. This evolution can be *event*-driven (by the occurrence of discrete instantaneous actions) or *time*-driven (by time elapsing) [28]. In the case of event-driven dynamics, it is only the occurrence of asynchronously generated discrete events that forces an instantaneous change of the system state (e.g., starting a motor or reaching a target position). For time-driven dynamics, the system's state continuously changes as time changes (e.g., a position and velocity values changing during a movement). In real-life ICSs, both event-driven and time-driven components are interacting for the operation of such systems. Taking the example of a positioning system, the position and velocity values follow time-driven dynamics, while changes in operating modes like "start" and "stop" of the movement are event-driven. In order to control industrial systems with hybrid dynamics, *continuous* and *sequential* process control are to be distinguished. Continuous control aims at maintaining a physical value at a desired setpoint typically through a Proportional Integral Derivative (PID) (Proportional–Integral–Derivative) controller. This is the well-known closed-loop control mechanism. Whereas in sequential control, sequences of commands are sent to switch the physical process from a state to another, therefore changing the context.

1.1.3 IT vs. OT

ICSs historically operated as isolated systems, with proprietary communication protocols. Therefore, they were considered protected from cyberthreats (the "security by isolation and obscurity" myth). This paradigm is far from reality and no longer supported. Due to the lack of security-by-design of legacy communication protocols, security concerns for ICSs have risen with the pervasive deployment of IT into the OT world, especially Internet-related technologies. Despite this increasing interconnection between IT and OT, ICSs have specific characteristics dependent on the context within which they are used. Such remaining

[1] https://www.as-interface.net/
[2] https://www.profibus.com/
[3] https://modbus.org/
[4] https://www.can-cia.org/

differences have to be taken into consideration when thinking about security. Consequently, traditional MISs and ICSs differ in the following ways:

Purpose of systems

ICSs stand out by the fact that they run physical processes (production units and chains, water and energy distribution networks, road and rail infrastructures, etc.); some of these also carry out protective functions for assets and individuals or for the environment [37]. As a consequence, in an ICS, the manipulated data has a physical meaning whereas it is not the case in a traditional MIS.

Performance requirements

The performance requirements in ICSs meet real-time constraints. This means that delays have to be controlled; high response times of signals and jitter are not acceptable. Such performance requirements can be explained by the fact that delays in data processing or transmission can deteriorate the performance and integrity of the physical process. On the contrary, MISs response times and delays are not time critical.

Lifespan

Whereas in the IT domain, the lifespan is of the order of 3 to 5 years, the OT world has an extremely long lifespan of more than 10 years (sometimes 30 or 40 years). The very fast lifecycle of IT components is due to rapid technological advances and the fact that compatibility is a technical requirement leading to homogeneity and interoperability. Furthermore, upgrades are straightforward with the availability of automated deployment tools. In contrast, it is more difficult to test the compatibility of components in ICSs mostly because of proprietary Operating Systems (OSs) (no homogeneity between components) and also due to the difficulty to interrupt an industrial system under operation. Hardware and software changes must be carefully made and even avoided when possible. Moreover, within industries, investments for OT are generally scarce with big financial expenses. This long lifespan results in an overlay of successive waves of technologies resulting in the phenomenon of equipment and software obsolescence.

Components

Most ICS components have limited computing capabilities, with tailored memory. Components follow very strict constraints on the response time needed by control functions; also, ICSs network communications are submitted to bounded transmission delays in order to ensure the timely realisation of control functions. Such time critically does not apply in the IT domain as delays are generally accepted especially with communication protocols for multimedia services. For example, requirements for transmission delay in Voice Over Internet Protocol is to be inferior to 400 ms (human hearing has a tolerance for speech delays of up to 250 ms) [69]. On the security aspect, ICSs' components are designed to support the physical process only and may not have enough memory and computing resources to support the addition of security capabilities. In opposition, IT-based systems are specified with enough resources to support the addition of third-party applications such as security solutions.

***Post mortem* incident analysis**

Following this idea, due to a unique and uncommon architecture, the reproducibility of an incident in ICSs is hard if not impossible. The number of parameters to compute is very large and the environment very complex. Additionally, depending on the application domain, some systems have to be able to maintain their activities (thanks to redundancies) or operate in degraded mode.

Physical environment

MISs show a climate-controlled environment such as a home or an office, whereas ICSs are deployed in a harsh environment with possible dust, high amplitudes of temperature, vibration, humidity, electromagnetism and/or proximity of dangerous materials. Generally, they are also distributed over geographical space on the scale of several kilometers. As an example, the Large Hadron Collider[5] which is a particle collider located beneath the France–Switzerland border is a 27 kilometers circumference industrial system. It has more than 1500 actuators and tens of thousands of sensors distributed along the structure.

From the above discussion, one can see that IT and OT present intrinsically different characteristics. A transposition of security measures of IT systems to OT systems would not be adequate. However, ICSs are the witness of a convergence between OT and IT in their composition. This is a rising concern, as ICSs were historically designed to operate in an isolated manner. At that time, a cybersecurity incident was not a risk worth considering. This has repercussions on current systems, mostly due to the very long lifespan of ICSs. Therefore, security concerns are more topical than ever.

1.2 Attacks on ICSs

From their typical architecture and the specific characteristics they meet as previously discussed, ICSs present a large attack surface. As a matter of fact, they are victims of a rising number of cyberattacks. The following Section gives the context of attacks targeting ICSs.

1.2.1 Definitions and Context

IEC 62443 standard gives the following definition for an attack: it is an "assault on a system that derives from an intelligent threat – i.e., an intelligent act that is a deliberate attempt (especially in the sense of a method or technique) to evade security services and violate the security policy of a system." [89]. To go a step further, an *intrusion* is the success of the attack into a system. Both attacks and intrusions stem from threats and vulnerabilities. If threats are external to the system, vulnerabilities are internal and can be studied and significantly reduced. A vulnerability is a security exposure that results from a weakness of the system that could allow an attacker to compromise its integrity, availability, or confidentiality [10]. The triad integrity – availability – confidentiality is the guiding principle of security within an information system, and is detailed in Section 1.3. To the question "Why are intrusions possible?", one could answer that any system has security breaches: vulnerabilities. Even if they can be reduced to a minimum, vulnerabilities are difficult to totally eradicate. Vulnerabilities have various origins, here are some common examples:

- Related to the components: software, hardware.
- Related to the overall system: specification, architecture, codes, etc.
- Related to the usage: configuration, administration, deployment, etc.
- Related to the user: password, best cybersecurity practices, etc.

From these definitions, the context in which ICSs operates can be set. Another piece of context concerns the measures taken against vulnerabilities. While in the IT domain, vulnerabilities are corrected through software patches and fixes are published regularly and easily deployed, industrial systems cannot employ the same protective measures. This can be explained firstly by limited memory and computing resources that cannot execute other tasks than process control relative ones, secondly by the great heterogeneity in the components and technologies composing an ICS which makes automatic deployments difficult and thirdly the

[5] https://home.cern/science/accelerators/large-hadron-collider

fear to disturb the normal functioning of the system. For instance, in a manufacturing system, it is generally highly undesirable and costly to interrupt production. Furthermore, in ICSs, cybersecurity concerns are quite recent (less than 15 years) compared to the IT world where awareness of cybersecurity risk is high. Even though, generic passwords (the well-known "admin admin") in industrial devices or unused ports kept open by default are becoming less common, vigilance is the watchword since the lack of awareness about ICS cybersecurity risks remains.

Consequently, it is not surprising to observe a rising number of attacks targeting industrial systems [91]. Of course, it is worth citing the well-known Stuxnet attack directed against Iranian Natanz Nuclear facility in 2010 [53]. This attack emphasized, for the first time, the need for ICS security. Stuxnet was the first attack with high media coverage. However, Stuxnet was not the first ICS attack, and security incidents were already reported years before. The first ones were identified around the years 1980 and 2000. At this time, attacks were sporadic and not sophisticated such as password change on a PLC[6], operation workstation sabotage by introducing a virus to sabotage a nuclear plant[7], wrong code downloaded on a PLC[8], etc. Although very impacting on the physical processes, reported cyberattacks at their beginnings remain simple. These attacks are generally initiated by insiders acting with malicious intent. They are carried out by opportunity and do not require sophisticated techniques. In 2000, the first sophisticated and remote attack occurred as an attacker successfully broke into a control system. It is known as the Maroochy Shire attack [1]. The attacker was able to use radio equipment to issue commands that lead to the spillage of 800,000 liters of raw sewage into local parks and rivers. In this cyber incident, the insider had a great level of familiarity and knowledge of the industrial system (since it was his previous job) which permitted to create the first sophisticated attack that ever occurred against ICSs. Then, in the next years, many different kinds of attacks emerged. Some attacks aim at propagating extensively and massively. These viruses or worms can end up disrupting an ICS even if they generally have not been designed to target a specific organization. For instance in 2005, the Zotob[9] worm used vulnerabilities on Windows 2000 version. The exploit massively scanned for vulnerable systems and impacted more than 700 companies. From 2010 (inherited from Stuxnet), industrial systems are specifically targeted by Advanced Persistent Threats (APTs) with a special attention given to PLCs. Emerging risks are posed by APTs since they are specialized types of attacks designed to exploit a specific organization. This highlights the rising complexity and sophistication of attacks in recent years. Supply chain attacks also have to be mentioned since they are also recently multiplying. These attacks consist in gaining access to the control system environment by means of infected products (hardware, firmware or software). Two cases can occur: (i) the manipulation of products is done before their delivering to the end consumer (namely the targeted organization) or (ii) the manipulation operates during the products' update process, via a corrupted update software for instance. Recently, the X_-Trader software supply chain attack affected two critical infrastructures in the energy sector, one in the U.S. and the other in Europe. The infected software was a voice and video calling software used by the two organizations[10]. Aforementioned attacks are shown in the timeline Figure 3. Two recent attacks, Poisoned Water and INDUSTROYER.V2 are further detailed in next Sections.

The techniques employed in cyberattacks have diversified (with technical advances) and become more complex. As a response to this phenomenon, a heavy classification work has

[6] *PLC Password Change*, Canada, 1988 https://www.risidata.com/Database/Detail/plc-password-change

[7] *Computer Sabotage at Ignalina Nuclear Power Plant*, Lithuania, 1992 https://www.risidata.com/Database/Detail/computer_sabotage_at_nuclear_power_plant

[8] *Wrong Code Downloaded to PLC Causes Plant to Shutdown*, United Kingdom, 1997 https://www.risidata.com/Database/Detail/Wrong_Code_Downloaded_to_PLC_Causes_Plant_to_Shutdown

[9] *Zotob worm*, Worldwide, 2005 https://www.f-secure.com/v-descs/zotob-a.shtml

[10] *Supply chain attack affecting Critical Infrastructure*, U.S. and Europe, 2023, https://symantec-enterprise-blogs.security.com/blogs/threat-intelligence/xtrader-3cx-supply-chain

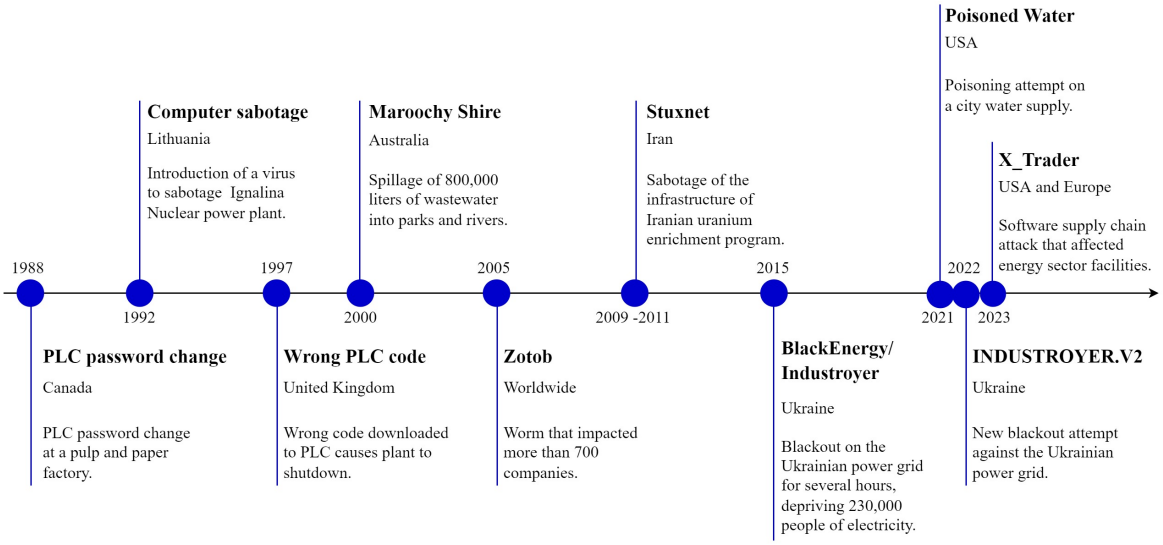

Figure 3: Chronological occurrence of some ICSs attacks since 1988

been done by the MITRE Corporation. MITRE, among other tools, created a Matrix for ICSs attacks. The MITRE ATT&CK (Adversarial Tactics, Techniques and Common Knowledge) Matrix [116] for ICSs is a framework for understanding and categorizing the various Tactics, Techniques and Procedures (TTPs) used by attackers during an ICS cyberattack. This Matrix is showed in Figure 4 and presents 12 tactics (columns) each of which encompasses several techniques. This MITRE framework helps to reflect the various phases of an attack from the initial access to the impact of the attack (each tactics from left to right in the matrix). This framework also provides mitigations and countermeasures. However, it is important to note that relying on the MITRE framework can present some limitations. A technique coverage is rarely complete as a technique can be broken down into several expressions. In other words, saying that a solution/product covers a technique does not mean that this solution/product detects all the operating modes associated with that technique. Furthermore, the MITRE framework would require frequent updates to tackle recent attacks. And this framework is by no means a security certification tool. All in all, this framework is a *de facto* reference for attack classification but has to be used wisely.

In the following, we are going to discuss recent attacks whose analysis is relevant for our work. We focus on attacks specifically tailored for ICSs and aiming to disrupt the physical process. Therefore, we will not detail any attacks that disrupt industrial systems by ricochet (e.g. ransomwares), as such attacks are not specifically dedicated to industrial systems. We are going to briefly detail an attack that was an attempt of poisoning a water treatment facility in Florida and an attack that resulted in a power outage in Ukraine. These examples will help to set the context and the deployment of an industrial attack targeting the physical process of an organization.

1.2.2 Poisoned Water

In 2021, Oldsmar water treatment facility in Florida was victim of a poisoning attempt in the city water supply [138]. This attack consisted in accessing remotely an operator computer of the water treatment facility. The cyberattack was instigated by the attacker using stolen credentials in order to access a machine running a remote access software package installed on it (called TeamViewer). The vulnerability partly originated from the fact that the operator machine had an outdated operation system no longer supported, and the fact that TeamViewer is accessible from the Internet since it is a remote control software used for maintenance purposes. Then, the attacker was able to manipulate the control setpoint for the dosing rate of sodium hydroxide into the water. This is a chemical often used in drinking water treatment

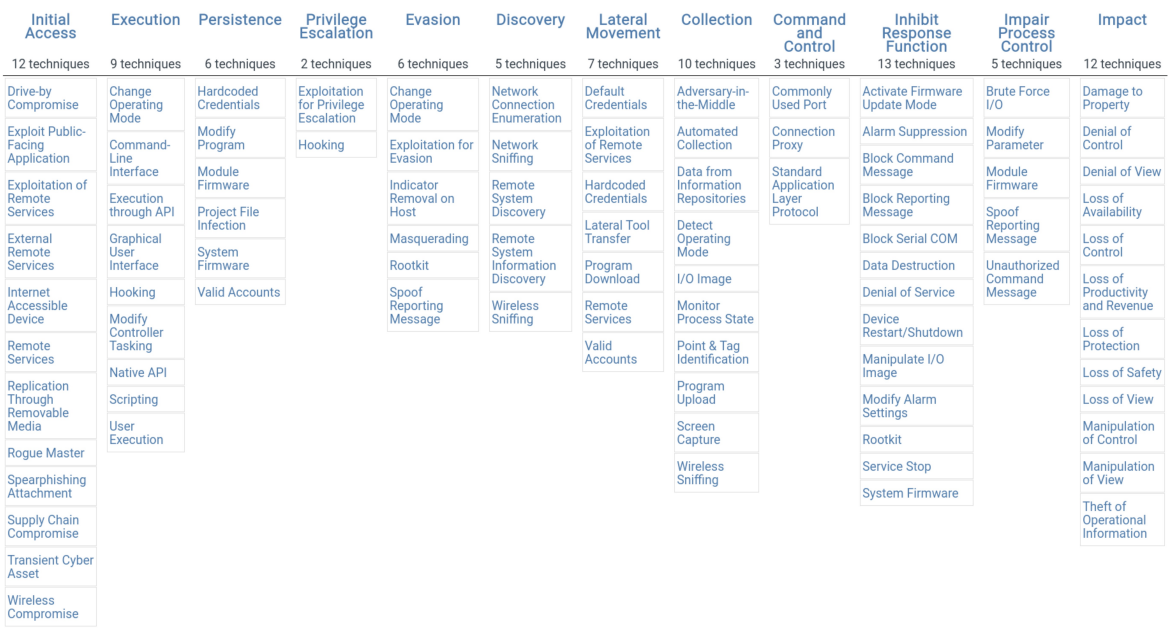

Figure 4: MITRE ATT&CK for ICS *© 2024 The MITRE Corporation. This work is reproduced and distributed with the permission of The MITRE Corporation.*

used to adjust pH and alkalinity. Although important as a treatment for drinkable water, this chemical can be dangerous for human beings if its concentration exists in excess into the water they drink. Here, the cyberattacker was able to increase the amount of sodium hydroxide into the water from 100 parts per million to 11,100 to poison the water supply. It was reported that the human operator observed on his screen the mouse moving, making changes, and exiting the system. The human operator immediately restored the normal operating parameters and the incident was closed.

This cyber incident shows that unsecured remote access can have serious consequences for the population. Remote access should be managed and architected safely; for example, best practices would determine which IP addresses are allowed in the network traffic, which communication types are allowed, and what processes can be monitored. Also, best practices should impose a limit on the setpoint value into a "safe" range of values. Organizations should also consider utilizing distinct networks, in order to isolate the control network to prevent such incidents. On the monitoring side, nothing can tell if an adequate system monitoring was deployed in this facility and could have detected the intrusion. Monitoring systems indeed helps to monitor key events within the control system including changes in setpoints of critical process parameters, changes in the control logic, etc. Luckily, the operator stopped the attack immediately. However, this incident emphasizes the importance of security measures, and an intrusion detection system approach **focused on process data** would have detected the intrusion with a very high probability.

1.2.3 INDUSTROYER.V2

The ongoing Russia-Ukraine conflict had resulted in an increase of cyberattacks in Ukraine, as well as in Russia. On April 2022, a cyberattack was reported against the Ukrainian power grid[11]. The attack leveraged different pieces of malware including a variant of Industroyer [144] (also referred to as CRASHOVERRIDE), a well-known malware that stroke Ukraine back in 2016, creating a blackout in the country for 6 hours. INDUSTROYER.V2, as its predecessor, is specifically targeting electric grids. This variant implements only the IEC 60870-5-104 (IEC-104) communications protocol which is used for power system monitoring and control over

[11] https://www.mandiant.com/resources/blog/industroyer-v2-old-malware-new-tricks

TCP/IP in Europe mainly (whereas its predecessor supported additional protocol modules to extend its reach to other geographical regions). However, the new version enables the embedding of customized configurations to adapt to specific Intelligent Electronic Devices (IEDs) making this attack extremely extensible and sophisticated.

The attack consists in: (i) an initial backdoor, (ii) the payload module, and (iii) a wiper to hide the attacker's activities. The initial backdoor is the entry point for the malware into the targeted system. It allows the attackers to gain access to the ICS network and enables sending and receiving commands from it. The payload implements the IEC-104 protocol in order to manipulate the ICS. It has the ability to: enumerate and create new processes, initiate communications and collect information, execute commands, manipulate files and services. Therefore, remote stations were targeted and actions involving opening and closing circuit breakers were performed. Then, a data wiper clears registry keys, erases files and kills processes to hide any suspicious activities. The data wiper also overwrites some files in order to make the system unbootable and harder to recover from the attack. Finally, the attack was successfully detected in progress and stopped before any actual blackout could be initiated.

Beyond implementing protective control measures such as network segmentation of the various sub-systems comprising the ICS, detecting such tailored attacks would require an intrusion detection approach able to monitor the state of sensors and actuators, and detect any deviation from normal behavior. Indeed, the data collected for detection means should be as close as possible to the physical process since the targeted equipment are IEDs. Additionally, **the multiplicity of targeted equipment in a geographically spread out environment calls for a distributed intrusion detection approach** that could handle local detection at certain critical local loops alongside a more global detection.

Through these few examples of recent ICS tailored attacks, one can witness the disastrous consequences of ICS attacks. They can be dramatic and far-reaching, including having an impact on the physical process, the financial aspect, the environment, interrupting essential services of a country (water, electricity, etc.) or even originating human costs. This highlights the clear need for cybersecurity solutions that considers the physical process of ICSs. Next Section sets the scene for ICS cybersecurity.

1.3 Security in ICSs

As previously discussed, the modern ICS is a wide range of OT and IT components. Despite this growing interconnection, the specific features of ICSs make the deployment of classical security controls (like anti-viruses or firewalls) not sufficient or even impossible (antivirus on a PLC would abolish real time, for instance). Security measures have to be adapted to the system they intend to protect and additional security controls have to be implemented. In order to deploy relevant security measures, it is important to understand the fundamental security properties of the system to be protected and the risk it encounters.

1.3.1 Concepts

Cybersecurity aims at analysing system vulnerabilities in order to deploy protective measures to limit them and be in a position to safeguard the continuity of core business functions to an acceptable extent [37]. In order to ensure good functioning, an industrial system respects some fundamental security properties[12] (in priority order):

Availability. It must always be possible for an authorized entity, user or process to access the system services. Operations intended to illegally take up processing time must be detected.

[12] Usually the triad Availability – Integrity – Confidentially is presented. However, sometimes other security objectives are added such as *Authenticity* or *Traceability*.

Integrity. The information must not be altered or destroyed, in an unauthorized manner. This refers to the protection of data sent by sensors as well as any commands issued into the system. The objective is to ensure that the data is preserved.

Confidentiality. The information has to be inaccessible to any unauthorized user, entity or process. In ICSs, the information concerns any sensitive parameters and data such as manufacturing formulae, quantities of substances used, data relative to performance or planning, PLCs or other equipment programs, devices configurations, passwords, and so on. These can be exploited by malicious actors in order to create a targeted attack.

Note that in the IT domain, there is a reversion in the priority order of these security attributes: Confidentiality is the highest priority and Availability is usually the lowest priority.

Other concepts are crucial to our work and need to be defined: the notions of *safety* and *security*. IEC 62443 standard defines the *safety* of a system as the property to be "free of unacceptable risk" [89]. *Security*, on the other hand, relates to risks from malevolent actions (e.g., cyberattacks) that can impact the system itself in addition to its environment [100]. Therefore, security considers potential threats due to attacks while safety considers hazards and faults.

Security and safety share common characteristics: even if the nature of the risk is different, the observed consequences may be the same especially on the physical process. Therefore, there is a strong interdependence of the effects of safety and security. To cite an example, in [77], the authors investigate how six different stakeholders in industrial automation address safety and security risks. In this case study, one third of the stakeholders were victim of a cybersecurity incident that led to a safety-related incident.

Every organization deals with risks every day. It is therefore necessary to work on risk management and assessment in order to provide sufficient knowledge and awareness as well as relevant control mechanisms with the general objective of minimizing the risks.

1.3.2 Risk Management

Risk management and assessment are processes defined in the ISO/IEC 27005 standard [83]. They are also described and developed in many national security organizations such as the NIST or ANSSI as guidelines for industries. The risk management process consists in addressing and attempting to anticipate and manage the effect of unfortunate events. Risk management process requires continuous efforts throughout system's lifetime. Risk assessment process is a subset of risk management. It focuses on detecting hazards and analysing all potential risks in a workplace.

Risk management can be decomposed into four steps: *framing*, *assessing*, *responding* and *monitoring*. These components apply to risk management for any risk related to information security, physical security, safety or finance [58].

Phase 1 – Framing. This phase consists in developing a framework for the risk management decisions to be made. These steps define how to conduct risk management activities and state the assumptions that has to be taken into account (including legal or financial constraints or expectations for example). In the case of ICSs, major considerations concern availability of services provided by the ICS, safety and security.

Phase 2 – Assessing. This phase consists in identifying critical resources to protect and identifying threats and vulnerabilities to these resources, with the underlying hypothesis from the framing phase. The analysis of the potential impact from an incident must incorporate the effect on the physical process, impact on dependent processes, and impact on the physical environment among other possibilities. For instance, an incident that would affect the ability to control or monitor operations could have a deep impact on the physical process by failing to provide commands when needed, sending incorrect commands, sending commands with

undesirable delays, etc. If this occurs, necessary response/control actions may not occur to prevent undesirable physical events from occurring. The final part of assessing risk is determining the likelihood that a particular incident will actually occur.

Phase 3 – Responding. This phase concerns defining a security policy. The security policy explicitly specifies rules that distinguish authorized from unauthorized behavior. This step defines who is authorized to use a resource and how, who is authorized to grant rights on the resource and who has administrator privileges.

Phase 4 – Monitoring. This phase is about maintaining an ongoing situational awareness about the security and privacy posture of the considered system and the organization in support of risk management decisions. This monitoring step measures the effectiveness of controls and environment changes that impact the security and privacy posture of the system.

Note that, in line with the principle of risk management, *threat modeling* can be carried out by organizations. A *threat model* defines which parts of the system are targeted, what is the goal of the attacker and which class of attacks are to be considered in order to deploy relevant controls.

1.3.3 Security Controls

The implementation of the security policy in a system, consists in selecting the set of *security controls* (or *countermeasures*) able to enforce the security policy. Three types of security controls have to be distinguished:

- *Preventive* security controls which try to prevent the occurrence of a security incident.
- *Detective* security controls which act when a security incident occurs.
- *Corrective* security controls which act after a security incident occurs.

IEC 27001 standard [81] sets a framework for an organization to establish, implement, maintain, and continually improve information security. The Appendix A of IEC 27001 standard lists security controls and IEC 27002 [82] standard provides a set of guidelines and best practices for implementing these security controls. This set of security controls can be used as a reference. Therefore, an organization should select and implement security controls based on their own risk assessments and needs. An organization might not need to implement every single security control but rather choose the ones most relevant to their situation.

The security controls provided in the standards, are categorized into the following themes: (i) Organizational controls, (ii) People controls, (iii) Physical controls and (iv) Technological controls. Each theme, encompass many controls. We are going to provide some examples for each theme.

Organizational controls (i) are related to establishing clear rules and guidelines for security, defining roles, responsibilities and accountabilities. It also concerns the classification and labelling of information, access control, but also compliance with policies, rules and standards. This theme also deals with information security in supplier relationships and information security incident management. Threat Intelligence is also concerned by this theme. This theme needs to be supported by appropriate technological controls such as logging, restrictions to privileged access, secure authentication, etc.

People controls (ii) are related to terms and conditions of employment, awareness, education and training. All in all, this theme addresses security concerns in employees lifecycle processes.

Physical controls (iii) deals with preventing any unauthorized physical access to sensitive locations. It also applies to the intentional introduction of new devices designed for malicious activities, or to unauthorized observations (visual observations, note taking, photographs, etc.). Note that physical access is often reported to be the initial access to a system in many cyber incidents. Physical security controls can be obtained by physical protection (cameras, sensors and access limiting systems) or by personnel security.

Technological controls (iv) concerns, among others: privileged access rights, secure authentication, protection against malware, logging, monitoring activities, network security, segregation of networks, use of cryptography, secure coding. Among all these security controls, the majority are *preventive*. For example, it is the case for segregation of networks. The perimeter of each domain should be well-defined. If access between network domains is allowed, it should be controlled at the perimeter using a gateway (e.g. firewall, filtering router). While it is important to separate the organization's network from the Internet, it is even more important to have firewalls between the ICS network and the rest of the organization's network. A good practice is to implement a Demilitarized Zone (DMZ) which is an arrangement of firewalls used to isolate a subnetwork from other networks. It works as a buffer zone.

Remaining security controls are *detective* and/or *corrective*. For instance, it is the case for monitoring activities. The purpose of this security control is to detect anomalous behaviour and potential security incidents. Monitoring activities are often conducted using specialized software, such as Intrusion Detection Systems. An IDS is a *Security service that monitors and analyzes system events for the purpose of finding, and providing real-time warning of attempts to access system resources in an unauthorized manner* [89]. This type of control is particularly interesting as it can operate in a completely passive manner to avoid disturbing the functioning of the ICS (note that active approaches exist as well). It seems to be a promising security measure, adapted to ICSs with possible detection at lower architecture levels without impacting the system.

From the aforementioned discussions, IT security policies may require changes to suit the needs of control systems. As stated previously, in contrast to IT, in industrial systems it is generally critical to provide immediate access to systems and applications. As an example, security policies that lockout user accounts after a certain number of failed passwords is completely unsuitable to ICSs. Security controls must not introduce time latency which might disturb the control loops. Many countermeasures are not compatible with this constraint, nor to the limited computing and memory resources of industrial components. Indeed, many security controls are only relevant for IT networks/components. Therefore, they can be deployed on high levels of ICSs, but few countermeasures focus on components and networks closest to the physical process, which are the most critical. Among the few methodologies that can tackle this aspect, stand out IDSs which are going to be the subject of this manuscript.

Conclusion

In this first Chapter, we introduced the typical architecture and characteristics of Industrial Control Systems. ICSs are complex systems, with hierarchical and distributed control functions, they encompass hybrid dynamics and they can be geographically spread out. From their typical features, industrial systems present vulnerabilities and a large attack surface. Many recent attacks showed the disastrous consequences they can have on physical processes. We pointed out how crucial it is to ensure the correct functioning of these critical infrastructures. It was discussed that due to the differences between IT and OT, most IT security countermeasures are not quite adapted for ICSs. From both the level of sophistication in

recent attacks, and the typical features of ICS components, we highlighted the value of intrusion detection systems, based on physical process monitoring. Additionally, we mentioned that distributed detection could tackle the spread out aspect of ICSs. In the next Chapter, we explore the state of the art of Intrusion Detection Systems within ICSs.

Chapter 2

State of the Art

Contents

Introduction

This chapter is dedicated to a state of the art related to our contributions on the intrusion detection task for ICSs. Section 2.1 defines and characterizes the concept of Intrusion Detection System (IDS) with a focus on anomaly-based IDS since this is the category of IDS we developed in our approach. As our work uses results from runtime verification for our detection approach, Section 2.2 gives an overview and state of the art of runtime verification for a better comprehension of our contributions. Finally, the topic of distributed intrusion detection in the literature is investigated in Section 2.3, as it constitutes another contribution of our work.

2.1 State of the Art of Intrusion Detection Systems

Intrusion Detection System was introduced in Chapter 1 as one possible security measure for an ICS. This Section reviews existing work on intrusion detection approaches.

2.1.1 General Concepts

IDSs were first introduced in 1987, in the work of Denning [42]. The concept was then systematized in the years 1999 and 2000, by Debar et al. who proposed a taxonomy [39], later revised in [41]. An IDS would present five distinct characteristics: *behavior on detection*, *detection paradigm*, *usage frequency*, *audit source location* and *detection method*. In the specific case of ICSs, some characteristics of this initial taxonomy are preferred than others. Some characteristics are still topical but almost not explored in the research field, leading to a consensus in the scientific community. For instance, the desired *behavior on detection* is usually preferred to be passive alerting for industrial systems. It means that the reaction of the IDS is to raise an alert without taking any active actions. Indeed, due to the nature of the manipulated data in ICSs, it could be extremely dangerous to take actions having a direct impact on the physical process. Furthermore, it becomes a new vulnerability for the system considering the fact that an attacker could trigger that active response on purpose in order to disrupt the system. Concerning the *detection paradigm*, the authors distinguish IDSs working on states and those working on transitions between states. The "states" mentioned are the following: "normal", "error" or "failure". This concept has been enriched by other authors and the detection mechanisms can be further detailed. Section 2.1.3 presents various detection mechanisms for behavior-based IDSs. Furthermore, the *usage frequency* concerns the monitoring capabilities; while some IDSs monitor in a real-time continuous way, others run periodically. On this aspect, the actual trend is to, as closely as possible, provide real-time information on the monitored system. Finally, the remaining two taxonomy dimensions are the more relevant for our research field and are discussed below. Industrial IDSs can be classified depending on the *audit source location* (sometimes called *data source*) and on their *detection method*.

A) Data source

An IDS can either be Host-based (Host-based Intrusion Detection System (HIDS)) or Network-based (NIDS). In the first case, the intrusion detection runs directly on host and it can access internal data of the host that it intends to protect. The second case captures network traffic without disturbing the communication. It requires the use of sensors in the network to catch traffic and analyze it.

An HIDS gives access to some data that is not visible into the network traffic such as internal variables or historical logs. However, an HIDS is extremely complicated to implement in the specific case of industrial environments. Indeed, as already discussed in the first Chapter, ICS components are usually embedding proprietary OS and are submitted to hard real-time constraints, they are therefore not including other software components than process control related ones. They present limited resources (computational power and memory) tailored for their industrial task and an HIDS would inevitably impact these resources.

A NIDS does not impact any equipment performance. And even if some of the internal variables of equipment are not accessible, a wider application area is available for NIDS. In other words, if an HIDS has a view only of its immediate surroundings, an NIDS has a view on every part of the network as soon as sensors are deployed. Packets can be captured after or before reaching an host which gives a better general view. For instance, analyzing packets on a fieldbus could be more efficient than monitoring host data in a PLC. Indeed, in addition to the difficulty of setting up such HIDS, its scope of action is restricted, whereas fieldbus monitoring is closer to the physical process and gives visibility of every equipment on the fieldbus.

In view of these characteristics, we directed our work towards NIDSs that seem a more adapted solution, nowadays, for the specific case of industrial systems.

B) Detection method

Two major detection approaches exist for IDSs: *knowledge-based* and *behavior-based* (also known respectively as *misuse-based* and *anomaly-based*). The first category uses the knowledge accumulated about attacks. They rely on specific recognition of malicious actions and are therefore able to detect patterns of previously known attacks. The second category consists in building a reference model of the usual behavior of the system being monitored. It relies on the definition of the "normal" behavior of a system and detects deviations from it.

The category of knowledge-based IDS stands alone in the IT world. Their main asset is that they generate few false positives, since the attacks are targeted in a precise manner. Yet their major disadvantage is that they are unable to detect unknown attacks.

Concerning behavior-based IDS, the main difficulty is to characterize the system's normal behavior. Generally there are two distinct phases: (i) the construction of the model, which represents the normal functioning of the system (ii) and the actual use of the model, consisting in comparing its information with data of the system during its execution. Creating an exhaustive model is a complicated task since the normal behavior can vary over time or over the system context of use. Therefore, behavioral methodologies usually generate a high number of false positives. But the real strength of these approaches is their ability to leverage the detection of unknown attacks. Various implementation techniques exist within behavior-based IDS to build the model, such as Machine Learning (ML), statistics, etc. Various approaches of the literature are discussed in Section 2.1.3.

A third detection method is identified by some authors as *Specification-based* intrusion detection [95, 135, 123]. Specification-based approaches combine the advantages of both knowledge and behavior-based detection techniques. As for behavior-based approaches, Specification-based methodologies use a set of specifications and constraints describing the correct operation of the system being monitored. But since specification-based detection techniques usually rely on manually developed specifications and constraints, they have a low false alarm rate compared to the high false alarm rate of behavior-based detection techniques. On the other hand, the cost to reach that low false alarm rate is that deriving specifications manually can be time consuming and hard to implement. Few work address this drawback and the state of the art is **lacking systematic methodologies for the specifications extraction**.

If some authors make a clear distinction between specification-based techniques and behavior-based ones, we prefer considering that specification-based approaches are a subcategory of behavior-based approaches as in [64, 113], since they share a common paradigm: characterizing the legitimate system behavior and raising alerts on deviations from a normal profile. Our work favors behavior-based detection and we direct towards specification-based intrusion detection. This choice of behavior-based detection is motivated by the necessity to detect unknown attacks in critical environments. Therefore such approaches have, according to us, potential to be very effective for intrusion detection in industrial systems.

2.1.2 Performance Evaluation

Performance evaluation aims at evaluating an approach through intrusion detection metrics or other key factors. Some measures, inherited from the IT world, exist for IDS performance evaluation:

- True Positive (TP) – Number of successfully detected abnormal activities.
- False Positives (FP) – Number of normal activities classified as abnormal. They are false alarms.
- False Negative (FN) – Number of abnormal activities classified as normal ones. They are undetected intrusions.

- True Negative (TN) – Number of normal activities that are successfully classified as normal.

Prediction \ Actual	**Positive**	**Negative**
Positive	True Positive (TP)	False Positive (FP)
Negative	False Negative (FN)	True Negative (TN)

Table 1: IDS verdict depending on prediction and actual activities

These metrics are synthesized in the confusion matrix depicted in Table 1. Using the four previously established metrics, some derived evaluation indicators can be defined:

- False Positive Rate (FPR) where $FPR = \frac{FP}{FP+TN}$
  Ratio between the number of normal activities detected as attacks and the total number of normal activities.

- False Negative Rate (FNR) where $FNR = \frac{FN}{TP+FN}$
  Ratio between the number of abnormal activities that are not detected and the total number of abnormal activities.

- True Positive Rate (TPR) where $TPR = DR = \frac{TP}{TP+FN}$
  Fraction of abnormal activities that are successfully identified. It is also called the Detection Rate (DR) or Recall.

- True Negative Rate (TNR) where $TNR = \frac{TN}{FP+TN}$
  Fraction of normal activities successfully classified as normal.

In the context of ML-based methodologies, the three following indicators are generally used: (i) Accuracy – fraction of activities that are correctly classified[1]; (ii) Precision – fraction of activities classified as true that are relevant[2]; (iii) F-measure (or F-score) – weighted harmonic mean of Precision and Recall[3].

In addition to these metrics, others can be found in the literature, such as Central Processing Unit (CPU) usage, detection decision velocity (or detection response time) which is the elapsed time between the launching of the attack and its actual detection. Also, Receiver Operating Characteristics (ROC) curves can be used to assess performance by visualizing the relation between TPR and FPR metrics.

It is important to note that these measures were historically applied for IT applications dealing with important volumes of data. There are less relevant for ICS-specific IDSs and authors in the literature arbitrary use few of them, if they use some at all. In [152], the authors emphasize the lack of a unified framework for IDS performance evaluation, despite the large quantity of research on the domain. This can be explained by the diversity of sectors IDSs are developed for (industrial systems, IT systems, theoretical studies, etc.) and the diversity of involved research communities (security, control theory, automatics, computer science, mathematics, etc.). In [106], the authors also point out that network-security oriented performance measures are not adapted to ICSs. According to the authors, packet-based ratios are less expressive than operational characteristics such as detection response time.

[1] $Accuracy = \frac{TP+TN}{TP+FP+FN+TN}$
[2] $Precision = \frac{TP}{TP+FP}$
[3] $F-measure = \frac{2}{\frac{1}{Precision}+\frac{1}{Recall}} = \frac{2TP}{2TP+FP+FN}$

2.1.3 Taxonomy of Behavior-based IDS for Industrial Systems

In subsection 2.1.1, we defined the major classification dimensions for IDSs. Due to the presence of the physical process, IDSs approaches can be further detailed with ICS specific characteristics, such as the nature and meaning of process data, the control logic executed by components, or the context of operation of the system, to mention a few.

As a matter of fact, various surveys propose to enrich the classification of IDSs with ICS-oriented taxonomies [160, 113, 159, 123]. In [160], the authors explicitly compare how SCADA's special needs are addressed in the reviewed work (they call this the "degree of SCADA-specific-ness"). This is an important characteristic because it permits to fully capture the characteristics of an industrial system with full situational awareness, including the system dynamic, its communication patterns, system architecture, and the data that is exchanged between equipment. In [113], the authors reviewed 28 articles. The survey discusses the differences between intrusion detection for IT systems and intrusion detection for ICSs. According to them, the main characteristics of ICS-oriented IDSs are the following: (i) Physical process monitoring, (ii) making use of the regularity and predictability of closed control loops, (iii) attack sophistication and extensive use of zero-day attacks, and (iv) dealing with legacy technology. Similarly, in the classification established in [159], the authors define general attributes to consider in ICSs. There are two main attributes: (i) the physical process, which is organized by the critical states of the system, and (ii) the intelligent control system. This latter, is built from multiple views: (a) communication (communication state, communication schedule, industrial network structure, and communication protocol), (b) task (task schedule and task state), (c) resource (network traffic, CPU utilization, and memory utilization), and (d) control data flow (timestamp and range of values). The authors in [123] address the case of specification-based IDSs for industrial systems. The taxonomy comprises: specification source, specification extraction, specification modelling, detection mechanism, detector placement and validation strategy.

Note that another way to further classify behavior-based IDSs is to categorize them by the methods used for building their detection model [64, 135, 93]. From these surveys, three main categories can be identified: (i) Statistics-based where the system is considered from a random viewpoint (using tools such as time series model, multivariate analysis, cumulative sums, etc.), (ii) (system) knowledge-based where the detection model is constructed with available system knowledge/data such as protocol specifications or network traffic instances (using tools such as FSMs or description languages, etc.), and (iii) ML-based where the model is constructed with samples of normal functioning of the system and allows the analyzed data to be categorized (using tools such as Bayesian networks, neural networks, fuzzy logic, clustering, etc.).

To review behavior-based IDSs from the literature, we propose a classification inspired by the previously presented taxonomies. We reviewed 39 papers; a detailed analysis is presented in Section 2.1.4 and all approaches are synthesized in Table 2. In the table, the approaches are presented alphabetically by author's names and they are classified according to the following characteristics:

- **Approach**
  This characteristic is inspired from the classification proposed in [159] to define the general knowledge of industrial systems. We argue that behavior-based detection requires at least a minimal process knowledge for its model construction. Therefore, this category concerns the part of the process that is taken into account for the detection model:

  1. **Communication between equipment**; This approach deals with the transmission of data into the ICS. In other words, this approach focuses on communication protocol dependant properties (such as the syntax and semantics) and communication flow (e.g. properties of packets belonging to the same flow, periodicity of

traffic, order in sequence of messages).

2. **Task and resources of equipment**; This approach deals with internal state of an equipment, its CPU utilization and memory utilization.
3. **Control data and control logic**; Control data concerns measurements status, i.e. data manipulated by actuators, sensors, but also data exchanged between controllers, supervisory control, HMIs, etc. Control logic represents the way commands are executed by controllers.

Among the different approaches identified, **Communication between equipment** and **Task and resources of equipment** correspond to the traditional distinction between NIDS and HIDS, whereas **Control data and control logic** is a wider concept that can encompass features of both communication and equipment (i.e. the transmission and the processing of data). The scopes of the three approaches are illustrated in Figure 5.

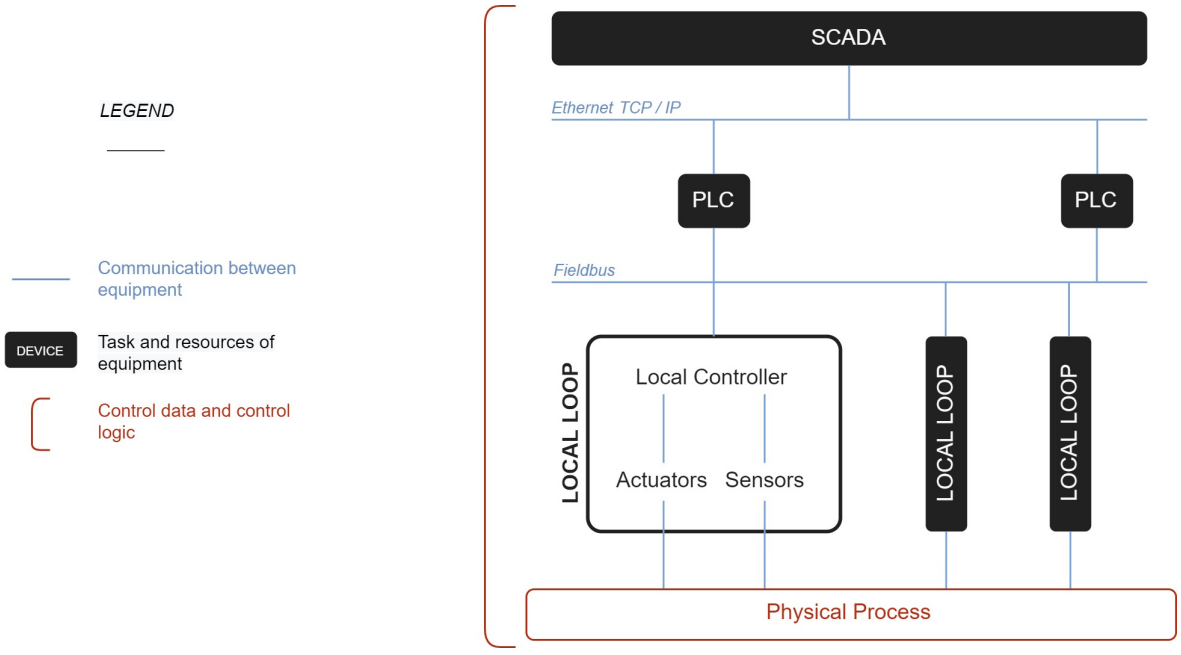

Figure 5: IDS approaches depending on the aspect of the industrial process they consider

- **Degree of physical process knowledge**
  It can go from "Low" to "High". This category is linked to the previous one (*Approach*) and, depending on what is modeled, different degrees of process knowledge is required. Consequently, modeling the communication between equipment, task or resources of equipment requires only a low degree of physical process knowledge. But taking into account control data and control logic shows a high knowledge of physical process knowledge.

  Intrusion detection approaches having a high level of physical process knowledge are often referred to as *process aware*. Similarly, we use the expression: *process aware attacks*. In the literature, the terms physics-based [67] and sequence attacks [27] appear and they are in the same vein. Both refer to the "physics" of an ICS or the semantics of the manipulated data. The authors in [27] divide sequence attacks into two different subsets: ordered-based and time-based. The former concerns attacks in which commands are sent with an incorrect order and the latter are attacks in which commands are sent with an incorrect timing. Hence, in both cases temporality of commands is implied in the malicious action.

- **Implementation technique**
  This category concerns the techniques that are used to build the detection model together with the formalism of the model.

- **Data source**
  This category gives precision of what type of data is used and its position in an industrial system. We distinguish network data source level which can be between supervisory and distributed control (i.e. between Level 2 and 1 of Purdue model, see Figure 1) and network data source at local loop level (i.e. between Level 1 and 0 of Purdue model).

- **Deployment**
  The deployment can either be *centralized*, *decentralized* or *distributed*. A centralized deployment refers to the concentration of processing and decision-making at a central point for the purpose of coordinating all the resources. In the context of IDSs, it means that even if the data source is multiple, the detection algorithm is unique and central. A decentralized deployment dispatches processing and decision-making across multiple points without any central concentration. In the context of IDS, it means that there are multiple detection algorithms, each establishing their own verdict. Finally, a distributed deployment also dispatches processing and decision-making across multiple points but with shared resources. In the context of IDS, it means that there are multiple detection algorithms that communicate and synchronize over a common network to achieve a common shared goal.

- **Hypothesis**
  This category states the hypothesis that are necessary to the approach.

- **Attack Typology**
  This category gives the type of attacks concerned by the approach.

- **Experiment**
  This category gives the experimental setup on which the approach was tested (physical testbed, simulation, etc.).

- **Evaluation Methodology**
  This category states the evaluation metrics used by the approach.

2.1.4 Related Work on ICS Behavior-based IDS

Since the IDS awareness level of the physical process is a driving notion for our work, we detail the previously introduced work by their **Approach**: (i) Communication between equipment, (ii) Task and resources of equipment, and (iii) Control data and control logic. There is also a Section dedicated to work combining multiple approaches.

A) Communication between equipment

Among the approaches about communication-oriented intrusion detection for ICSs, various aspects of the communication between equipment are considered:

Communication protocol

Communication protocol-oriented approaches are interested in protocol dependant properties such as the syntax or semantics of a communication protocol. The syntax defines how the data is structured, in other words, the order in which pieces of information are packaged. Whereas the semantics determine the meaning of individual pieces of information.

For instance, the authors in [29] were among the first authors to highlight the importance of using model-based approaches to leverage the detection of unknown attacks. They proposed a

rule-based model which constraints the values taken by the fields of a Modbus message. The model uses expected characteristics of Modbus TCP protocol by specifying function codes supported by a device, permitted lengths of certain data fields, exception codes and protocol identifiers. The work in [7] adopts a similar approach by opening the methodology to several protocols: DNP3, IEEE C37.118 and IEC 61850. The approach aims at detecting network packets that violate the formal protocol specifications, in other words the methodology is to filter invalid and malformed packets. The approach permits to detect efficiently buffer overflow attacks or any attack using invalid commands. Furthermore, it adopts a distributed methodology in order to reduce to a maximum network overhead during operations. However, these previously discussed detection approaches [29, 7] are limited to individual and isolated network packets.

In response to this issue, other works go further by building models of message exchanges. For instance the approach described in [68] relies on a Deterministic Finite Automaton (DFA) representing an HMI-PLC communication pattern. The DFA is automatically constructed based on about 100 messages, and characterizes Modbus exchanges. Two relevant Modbus fields are used to build the model: (i) the transaction identifier – a two-byte integer that pairs request (from the master) and response (from the slave) corresponding to a unique transaction and (ii) the unit identifier – a single-bit integer that identifies the Modbus slave associated with the transaction. This approach relies on the assumption that industrial network traffic is highly periodic and present fixed and repeated patterns. However, experimentation and future works showed that such a highly periodic hypothesis is closer to myth than reality and this kind of approach suffers from many false positives. In [94], the authors address this issue which can be explained by the *multiplexed ICS stream*. In ICSs, it is typical to observe a device sending multiple signals, over a communication medium, at the same time in the form of a single, complex signal. For instance, it is the case for read and write commands on a range of process variables, issued on a single network packet. Consequently, the authors in [94] propose to use multiple DFAs ("one per cyclic pattern") instead of one "very large DFA". Yet, this approach is more efficient when the patterns of multiplexed cycles are known, i.e. expert knowledge is recommended for the model construction.

All in all, deterministic approaches suffer from a substantial false positives rate and they respond to strong hypothesis such as high periodicity and regularity of industrial network traffic. Furthermore, they are not resilient to manual interventions of human operators – typically detected as false positives.

In response to deterministic approaches drawbacks, some other works focused on probabilistic approaches. Probabilistic methodologies permit to better deal with uncertainties in industrial network traffic. In [27], the authors rely on Discrete-Time Markov Chains (DTMCs) to model sequences of protocol messages. The approach aims at detecting sequence attacks and therefore built the model according to an ordered list of events. In this approach, the events consist of tuples with relevant fields and properties from network packets. In the DTMC, a state aggregates events with similar semantic meaning (e.g. a state that gathers together all the "write" commands that change a specific variable), whereas transitions between two states indicate order relations between them (e.g. an event from state B comes before an event from state A, in the sequence under analysis). During the training phase, probability functions are computed for states and transitions. During the detection phase, the current state is identified as well as transitions and their related probabilities from network traffic observations. Alerts are triggered by unknown states, unknown transitions or unknown transition probabilities. Based on dataset experiments, the authors found a high number of unknown transitions, causing false positives, that can be partly explained by network delays. The authors also propose to define "importance" in some specific events such as write operations that would be favored by attackers. This would allow to direct the detection on important aspects by adapting thresholds. The work in [55] extends the previous approach with a substantial model reduction. The authors change the generation algorithm, allowing to

combine states with overlapping information without impacting the accuracy of the models. This results in smaller models with lower computation times. However, this approach still suffers from a high number of false positives.

Similarly, the work in [157] uses Probabilistic Suffix Trees (PSTs) to model message sequences. An event is considered an anomaly if the probability of occurrence of this message, given a history of previous messages, is below a threshold. The main subtlety in this methodology is to consider multiple previous messages, whereas the approach presented in [27] only considered the previous message. The approach in [157] is therefore more resilient to noise, i.e. legitimate variations from the model traffic pattern, due to missing, out-of-order and/or sporadic events observed in industrial networks.

Another probabilistic approach is the one presented in [75]. The authors also mention the excessive number of false positives, they mention "hundreds or thousands, daily in [their] experiments". To overcome this drawback, they rely on multiple models for the detection. In other words, multiple Deterministic Probabilistic Automatons (DPAs) are built from traffic observations, corresponding to variations of a "conversation" between two ICS equipment. The model relies on information extracted from communication protocols, in particular packet headers containing message types and other relevant attributes. Building multiple models allows to capture subtle differences due to spontaneous events in the network traffic. Therefore, this approach addresses the issue of noise in industrial network traffic. Additionally, this approach incorporates delayed or re-transmitted packets into the training dataset to ensure the model is able to cope with such events without raising an alert.

As a matter of fact, most deterministic and probabilistic protocol-approaches rely on the heavy hypothesis of periodicity in network traffic. Furthermore, these approaches generally do not take into account the meaning of the network packet payload (i.e. physical process meaning): only the type of message is considered (such as read or write) and not the associated data. This can be explained by the necessity to keep state-models manageable, since considering every process variable could lead to an explosion of the number of states.

Communication flow

Flow-based approaches focus on the structure of exchanges between equipment within an ICS. The detection relies on observation of properties of packets belonging to the same flow, traffic periodicity and traffic regularity.

In the approach described in [5], the authors present a detection framework composed of three modules relying on different methodologies. One of them relies on a "rule-based flow behavior analysis", which uses ML to determine network traffic flow properties. Such rules are efficient against flooding attacks for instance, as they modify the usual regularity of network traffic.

The authors in [14] also propose a flow-based methodology. They define "periodic bursts" as a fixed number of packets being transmitted at fixed intervals. The approach therefore relies on period (or frequency) and size (number of packets) of these periodic bursts. The methodology consists in aggregating meaningful packets to create network flows (using server transport port, for instance) and to translate them into time series. That is, for every fixed interval, the number of packets belonging to a specific flow is stored and constitutes a "frequency fingerprint". This model is then used for online detection using signal processing techniques, namely Fourier transforms. The same authors went further in [15] by identifying different levels of periodic patterns. They define two types of connections: long-lived and short-lived network connections. Long-lived connections can persist for several minutes or hours, whereas short-lived ones are below few seconds. Compared to the previous work, the learned model encompasses a whitelist of valid commands together with the frequencies at which they are sent.

The work in [103] also characterizes ICS traffic periodicity and specifically focuses on

spontaneous events, i.e. traffic caused by non-polling mechanisms. The methodology relies on a PST, to model the diverse timing-patterns of spontaneous events. To do so, inter-arrival times are calculated from spontaneous events timestamps. These inter-arrival times are then categorized and used to build the PST. This work is extended in [104] with more experiments and datasets comprising new protocols (S7, S7-plus[4], MMS[5]). The same method is applied to observe communication flows, but this work addresses more generally server-driven traffic and their periodicity. The authors define server-driven traffic as the "traffic initiated by the servers in the client–server architecture without a paired request". The approach identifies server-driven packets and extracts from them relevant information to group them by flows. For each flow, timestamps of server-driven event arrival times are categorized and transformed into time series by calculating the number of server-driven events per some configurable interval of time. From these time series, a Multivariate Correlation Anomaly Detection (MCAD) approach is proposed and the output is a distribution of Generalized Variance (GV) of these time series. The idea is that the correlation measures and the inferred dispersion of data points from different flows (i.e., traffic from different monitored points) should stay within an expected range in a normal situation. If it is not the case, an alert is raised.

Flow-based approaches generally do not contain information on the physical process of industrial systems and periodic activities are learned, based on their periodicity. Consequently, when an alert is raised, little insight is provided to explain it. Moreover, an attack that would respect periodicity of traffic would appear as a legitimate flow leading to a false negative.

B) Tasks and resources of equipment

This category of approaches deals with internal state of an equipment, its CPU utilization and memory utilization. Few research work were conducted on this aspect due to the scarce computing resources and limited memory of components in industrial systems. Furthermore, instrumenting a component without impacting the real-time performance requirements of control loops is extremely challenging.

One area of development focuses on PLCs. The authors in [110, 111] deploy a rule-based model using temporal logic for offline PLC program check. Therefore, security constraints are checked at local loop level, before execution of the system. This approach relies on the strong assumption that the command law is correctly programmed into the PLC. During the online execution of the system, this approach also encompasses a rule-based monitoring of sensor values from physical observations, and control signals sent to actuators. For online execution, the model uses a state space language. The approach is able to actively react to anomalies: PLC code is instrumented (using Instruction List IEC standard). Instrumenting PLC code can lead to an increase in the scan cycle time. However, the authors proved that in the case of a denied operation, the scan cycle only presents a 1% overhead.

Another work [63] proposes an approach instrumenting PLCs. The approach uses PLCs coupled with embedded hypervisors. This way, the online detection is directly enforced within the PLC. The main advantage of using an hypervisor lies in the fact it runs an operating system with much more computation power than the actual PLC. The authors propose to develop a protection mechanism, integrated into the scan cycle of the PLC. The idea is to prevent PLC's clients to write in some memory zones before verification of the value of corresponding variables. To do so, the writes are restricted to a temporary buffer zone in memory, the written values are then verified and forwarded to the destination memory buffer if no safety or security constraints are violated. Such a methodology is hard to implement in real life industrial systems because deploying an approach into a resource-constrained equipment is a

[4] S7 and S7-plus are Siemens proprietary protocols.
[5] MMS (Manufacturing Message Specification) is a protocol defined by ISO 9506 and used in IEC 61850.

difficult task and inevitably impacts real-time performances.

Some approaches take a slightly different angle and enlarge tasks and resource-oriented detection to SCADA level. In [119], the approach uses process mining to build a Petri net model from device logs stored in a historian. The historian is a server, responsible for storing and logging all of the data that the SCADA system aggregates. And more recently in [90], the authors focus on SCADA Application programming interface (API) calls. A graph model is created where each state is a SCADA phase allowing specific API calls. The detection methodology operates via an hypervisor. This work relies on the assumption that the hypervisor is trusted and cannot be compromised. So far, the approach is adapted for Open Platform Communications (OPC)-based ICS deployment only and Windows-based SCADA. The novelty of this work lies in the fact it considers SCADA execution context and identifies execution phase-specific behaviors.

C) Control data and control logic

As discussed above, communication-oriented and tasks/resource-oriented detection are ineffective against stealthy and sophisticated attacks using deep knowledge of physical processes. Indeed, some attacks impact the physical process without inducing communication-related nor equipment's tasks/resources-related anomalies. Consequently, some approaches incorporate deeper physical process knowledge to develop their model. Process-oriented detection approaches generally rely on the monitoring of control loop events such as the evolution of sensors and actuators states. Monitoring measurable variables permits to estimate if the system is converging towards some critical state.

Some approaches focus on modeling the system's dynamic. The idea is to model physical properties of the system using laws of physics in corresponding domains such as fluid dynamics, electromagnetics, etc. Generally, the system is described through differential equations, forming a model to predict the future outputs of ICS at local loop level [127, 152, 92, 80, 112]. Equivalent approach works in the frequency domain as well, instead of the time domain, using Laplace transforms instead of algebraic equations. Therefore, a system can also be described with transfer functions [126]. These approaches come from control systems diagnostic theory, they rely on the assumption that the entire system can be described with differential equations which is possible only for time-invariant systems (i.e. which does not depend on time). In control theory, a time-invariant system has a time-dependent system function that is not a direct function of time. Depending on the physical process, it may not be possible to model the system's dynamic. Although encompassing deep knowledge of ICSs physical process in their model, these approaches are difficult to deploy as it can be challenging to collect data at sensors/actuators level. Furthermore, these approaches are not suitable for network detection as the models may require access to internal variables of controllers which are not available from the network traffic. Therefore, they would tend towards host-based intrusion detection approaches.

Another common approach is to test if sensor reading and actuators settings are following a prediction model, learned from network traffic. Therefore, the common methodology is to survey some memory variables of a PLC and reconstruct the image of process variables. The approaches in [73, 50] built auto-regressive models allowing to detect deviations and predict a range for future values. In [73], process variables are automatically classified into three categories: (i) continuous variables (such as sensor measures), (ii) discrete variables which can take a finite set of values (control program's states), and (iii) constants (such as configuration parameters). Then, a model is associated to every variable depending on its type in order to predict its next value. The work from [50] follows the same direction of research. PLC registers are classified into three classes: sensor registers, counter registers and

constant registers. Then, the detection is able to raise alerts when registers' values deviate from the learned models. These approaches does not consider correlation between variables which may be the case for variables in real life systems. Furthermore, this type of models reacts poorly to external disturbances that they would detect as deviations from standard behaviors.

Other approaches focus on critical process variables while relying on learning-based methodologies [62, 156]. In [62], the authors propose a neural network-based NIDS that classifies network traffic, including input features derived from physical knowledge of the system. This supervised learning approach takes as input a set of process variables to consider. This step requires expert knowledge on the physical process to determine the set of variables to consider. In [156], the authors introduce a SIMPLE IDS model. If SIMPLE stands for "Sufficient, Independent, Meaningful, Portable, Local & Efficient", it also emphasizes the desire to propose a simple detection approach, in contrast with recent literature works tending to be more and more complex. *Sufficient* refers to the ability of SIMPLE approach to detect most attacks while emitting few false alarms (compared to complex approaches). The approach is *independent* of parameters or specialist's knowledge, when a trained-model is re-evaluated or modified (indeed, especially in the context of ML, hyperparameters can influence the training process). *Meaningful* refers to the alerts that should permit to take appropriate measures in a timely manner. The approach is *portable* to different ICS processes. The approach is *local* in the sense that it is adjusted to particular sensors/actuators and their specific behaviors. To finish, *efficient* refers to the computational process required for training and detection phases. In this work, the authors focus on sensor and actuators values. They propose four independent modules: (i) MinMax approach which consists in detecting if actuators/sensors values got out of fixed limits, (ii) Gradient Approach which consists in detecting abrupt change in actuators/sensors values (i.e. speed of change), (iii) Steadytime approach which focuses on the time sensors/actuators do not change their values, and (iv) Histogram approach which focuses on occurrence of values through their distribution in a fixed time window. The authors prove their approach to be efficient against many attacks. However, learning-based approaches focusing on critical process values [62, 156] first need to identify the values of interest, which can be very challenging.

At the difference of the above methodologies, another line of work manually specifies their system for critical state detection. To do so, they identify forbidden states or allowed behavior within the system, using *a priori* knowledge of the system to build the model. In [59, 60], the authors develop a state-based IDS and they use systems' specifications to extract security properties for discrete and continuous variables. The state of the physical process corresponds to actuators/sensors values. The authors take the assumption that the critical states (i.e. the set of system configurations which damage the system) are well known, and they attempt to fully characterize the critical states. Then, the detection aims at predicting if the current evolution of the system will reach a critical state or not. The notion of distance from the current state to a set of possible critical states is defined to express criticality. The current state is obtained during the execution of the system by inspecting the content of network packets and retrieving relevant information. Another complementary approach is the one described in [26]. The approach also relies on monitoring the evolution of the process states and tracks down when the process is entering a critical state. In this work, the representation of critical states is made through a language called Industrial State Modeling Language (ISML). This language allows to express a *condition* leading to an *action*. A condition is a boolean formula (composed of conjunctions of predicates) describing values taken by process variables present in PLC registers; an action is a log (i.e. an alert). The conditions remain simple with no use of temporality specifications such as order of events or timing in the occurrence of events. In the same vein, [115] developed a similar distance-based approach for medical systems. The behavior rules are transformed to a FSM with three type of states: safe, warning and unsafe

states. The approach is further refined for discrete systems only, in [142]. A FSM is also constructed and each state represents the state of actuators and sensors. In this approach, a risk analysis is a prerequisite and highlights the feared events for the system (prohibited states list). Based on this, all the states are created and critical states are identified. The distance-based approach is enriched with the notion of trajectory which studies the evolution of distance through time. And a novelty in this approach is the implementation of temporal constraints; a characteristic execution time is assigned to each action (i.e a command and its consequence) and is used in the detection model. Concerning the detection during the execution of the system, a "filter-based" approach is used. There are two filters: (i) a control filter that checks orders exchanged between the PLC and actuators and (ii) the reports filter that verifies data sent from sensors to the PLC. The two filters cooperate and exchange information in order to feed detection mechanisms. If most approaches consider attacks on the action chain only (i.e. on actuators) taking as main hypothesis confidence in the data provided by sensors, it is not the case in this approach, because of these two filters. Finally, the approach is situated at local loop level (between PLC and sensors/actuators), but the experimentation relies on a simulation which makes data retrieval simple. Furthermore, simulated data may lack fidelity compared to real ICS data.

In graph-based representations (such as FSM, as in [115, 142]), where states are defined by combinations of sensors and actuators values, scalability is a critical issue. Indeed, for large-scale systems, considering every process variables would lead to an explosion of the number of states. The modeling phase would present an exponential complexity as the model describes all combinations of sensors and actuators states. A solution would be to sacrifice exhaustiveness and not consider every process variables. But the choice of which variables to consider would be arbitrary and dependant of a human expert, possibly leading to errors, false positives or false negatives.

The above approaches require, for the model construction, knowledge about the entire set of critical states. This requirement may be difficult to satisfy on large scale systems, and the exhaustiveness cannot be guaranteed. As a response to this issue, some work focused on automating the specification extraction phase.

The main drawback of specification extraction, is the manual aspect implied in the rule generation process. The set of rules depends on expert knowledge (making the model, assumption-based). However, some approaches are emerging with automated specification extraction. This is the case in [139], where the authors propose to automatically derive rules from IoT device's operational profile specifications. The operation profiles define the mission statement of specific equipment, allowing to derive the security requirements for this equipment. In [124], the authors follow the same direction by showing that partial protocol FSMs can be automatically extracted from Requests from Comments (RFCs) written in textual format. In a similar manner, the authors in [57] automatically translate common RTU configuration files into Zeek[6] scripts in order to deploy rules to monitor IEC-104 traffic. Note that Zeek is an open source IDS that allows sophisticated custom scripting. This allows to automatically generate dynamic rules, encompassing requirements on process values while adding simple temporal constraints.

The majority of the aforementioned approaches does not encompass time requirements into their detection model. For the few who does [142, 57], the rules remain simple and without correlating variables, such as holding an event for a specific time, for instance. Characterizing more complex time properties of ICSs would allow to better capture the dynamic of such systems. Adding the notion of *time* into a model allows to monitor actuators/sensors states over temporal behaviors. One could monitor the occurrence order of several events (more than two) together with timing properties (e.g., "After the occurrence of Event A, Event B has to occur in the next 10 seconds, whereupon Event C has to hold between range of values,

[6] https://zeek.org/

for a maximum of 30 seconds").

Consequently, another line of work chooses to use more expressive languages such as Temporal Logic (TL) languages. TL is a common family of specification languages that can be used to specify temporal patterns over the behavior of a system. Among TL specification languages, there are various formalisms with different levels of expressiveness, such as Linear Temporal Logic (LTL), Metric Temporal Logic (MTL) or Signal Temporal Logic (STL) for instance. Hence, in [54], the authors propose a specification-based IDS relying on specifications inspired from LTL. The model is built by observing the values of process variables (sensors and actuators values) over time in order to obtain temporal patterns. This process is automated but requires insight from a human expert to tune the model. The authors in [120] adopt the same methodology and propose a temporal rule-based model. At the difference of the previous approach, the model construction is entirely learned-based. Hence, identification of critical process values and extraction of time patterns from normal network traffic is automated. As a consequence, this approach is adapted for recurrent variables with critical values only.

Similar approaches, developed the methodology for the Internet of Things [71, 150]. Both approaches rely on a rule-based model using MTL. These approaches are adapted to discrete-state, event-driven systems only and they require expert knowledge to establish the set of specifications.

The work presented in [98] also emphasizes the need to leverage the temporal characteristics of system states. However, the approach goes a step further than the previous ones. As in [120], the specifications are also extracted automatically from network traffic using a specification mining algorithm. The detection level is based on the observation of network traffic between HMIs and PLCs. The detection requires knowledge of PLC programs and is limited to sequential programs. Hence, the notion of *activity* is introduced, which captures the different subprocesses and functioning modes of a sequential system. For instance, using activities allow to distinguish between manual or automatic modes of functioning, or between different phases such as start phase or shutdown phase of an operation system. In this approach, the activities are recognized from PLC control logic expressed as a Sequential Function Chart (SFC). These activities are used as context to delimit the rules, and sets of rules are assigned to certain activities. Furthermore, it gives insight on the actual state of PLC program relying on network traffic observations, without requiring physical access to the PLC. The generated rules are expressed through LTL and MTL properties.

Authors	Year	Approach	Physical Process Knowledge	Implementation Technique	Data Source	Deployment	Hypothesis	Attack Typology	Experiment	Evaluation Methodology
Y. Al-Nashif et al. [5]	2008	Communication between equipment	Low	ML; supervised learning	Network: traffic between Supervisory Control and Distributed Control	Centralized	Periodicity and regularity of industrial network traffic	IT-oriented attacks (Scanning, R2L, DoS attacks, worms); Does not work on attacks using "legal" commands	Testbed	Detection Rate (DR)
P. Anantharaman et al. [7]	2022	Communication between equipment	Low	Rule-based model; communication protocol semantics and syntax	Network: traffic between Supervisory Control and Distributed Control	Distributed	Correctness of the deployed rules (in accordance with protocol standards)	Attack that do not respect syntax of communication protocols	Dataset	Correctness of security parser, detection, ability to handle high traffic rates
R. R. R. Barbosa et al. [14, 15]	2016	Communication between equipment	Low	Signal Treatment; time series model	Network: traffic between Supervisory Control and Distributed Control	Non Available (N.A.)	Periodicity and regularity of industrial network traffic	Attacks that are not respecting periodicity, size or number of exchanged packets	Dataset from real life ICS	N.A.
A. Baláž et al. [12]	2010	Communication between equipment	Low	Statistical approach	Network	Distributed; two layers: (i) basic processing and (ii) central evaluation module	Dedicated to communication technology (does not specifically focus on ICS)	IT-oriented attacks	Testbed	Efficiency, acceleration and overbalance
A. Carcano et al. [26]	2011	Control data and control logic	High	Rule-based model for critical state construction; ISML (Industrial State Modeling Language)	Network: traffic between Supervisory Control and Distributed Control	Centralized	Knowledge of the entire set of critical states of the system	Attacks manipulating actuators (orders coming from Supervisory control or HMIs)	Testbed (3 PLCs)	FP, FN, DR, time elapsed for distance calculation
M. Caselli et al. [27]	2015	Communication between equipment	Medium	Discrete-time Markov Chain (DTMCs)	Network: traffic between Supervisory Control and Distributed Control	N.A.	No attack during learning phase; Requires a large volume of data; Periodicity and regularity of industrial network traffic	Sequence attacks (playing with order and timing of commands)	Real system (water treatment facility)	TPR, FPR
S. Cheung et al. [29]	2007	Communication between equipment	Low	Bayesian network + Rule-based model ;	Network: traffic between Supervisory Control and Distributed Control	Centralized	Protocol dependant (Modbus/TCP); Periodicity and regularity of industrial network traffic	IT-oriented attacks, attacks that are not respecting the standard communication protocol structure	Testbed (Sandia National Laboratories)	Analyze on the number of rules
J. J. Chromik et al. [31]	2018	Control data and control logic	High	Rule-based model on process variables	Network: traffic between Supervisory Control and Distributed Control	Decentralized	Expert knowledge for the set of rules	Process aware attacks	Dataset	Detection
N. Erez et al. [50]	2015	Control data and control logic	Medium	Automatic classifier for SCADA control register values (three classes)	Network: traffic between Supervisory Control and Distributed Control	Centralized	Periodicity and regularity of industrial network traffic; Expert knowledge for system's variables used in the classes	Attacks targeting variables from the training set	Dataset composed of 131 hours of SCADA traffic (7 PLCs, 449 registers)	Accuracy, false alarm rate

D. Fauri et al. [54]	2017	Control data and control logic	High	Rule-based model; Specifications inspired from LTL	Network: traffic between Supervisory Control and Distributed Control	Centralized	Knowledge about critical states/variables to watch	Attacks manipulating actuators (orders coming from Supervisory crontrol or HMIs)	HIL with 1 functional chain only	N.A.
I. N. Fovino et al. [59, 60]	2010 – 2012	Control data and control logic + Communication between equipment	High	Rule-based model for critical state detection	Network: traffic between Supervisory Control and Distributed Control	Centralized in a module for a firewall	Protocol dependant (Modbus/TCP, DNP3); Knowledge about the entire set of critical states	Single packet detection based on protocol specifications; Process aware attacks	Testbed (6 PLCs)	Number of raised alerts, expected alerts
B. Ferling et al. [55]	2018	Communication between equipment	Low	Discrete time markov chains	Network: traffic between Supervisory Control and Distributed Control	Centralized	Periodicity and regularity of industrial network traffic	Sequence-attacks	Dataset (10 days of traffic of a gas facility)	False positives, detection accuracy
R. Flosbach et al. [57]	2019	Control data and control logic	High	Rule-based model; specifications extracted from RTU configuration files and converted into Zeek scripts	Network: traffic between Supervisory Control and Distributed Control	N.A.	Expert knowledge needed to characterize the system (description of physical topology)	Process aware	SCADA traffic of distribution substation	False positives, false negatives
N. Goldenberg et al. [68]	2013	Communication between equipment	Low	DFAs	Network: traffic between Supervisory Control and Distributed Control	N.A.	Protocol dependant (Modbus/TCP); Periodicity and regularity of industrial network traffic	Attacks that are not respecting usual protocol traffic pattern	Reel system (campus power-grid)	Pourcentage of different types of transitions
W. Gao et al. [62]	2010	Control data and control logic	Medium	Neural Network backpropagation; (supervised)	Network: traffic between Supervisory Control and Distributed Control	N.A.	Periodicity and regularity of industrial network traffic; Expert knowledge needed about the physical process.	IT-oriented attacks; Process aware attacks targeting variables from the training set	Testbed	FPR, FNR, accuracy
L. Garcia et al. [63]	2016	Tasks and resources of equipment	Low	DFAs	Host: PLC + traffic entering PLC (PLC coupled with embedded hypervisor)	N.A.	PLC not corrupted; Periodicity and regularity of industrial network traffic	Attacks that modify PLC registers while respecting structure and periodicity of communication	Testbed	N.A.
M. Grochowski et al. [71]	2019	Control data and control logic	High	Rule-based model using Metric Temporal Logic (MTL); Runtime verification	Network: AWS IoT Core ; IoT cloud technology (Message Queuing Telemetry Transport (MQTT)) messages sent in cloud)	Centralized	Adapted to discrete-state, event-driven systems only; Temporal formulas given during the operating phase of the system (for the set of specifications)	Process aware	Cleaner robot (a proximity sensor, a camera, a pneumatic actuator and a robot arm)	Detection response time (time beween occurrence of events and processing)
D. Hadziosmanovic et al. [73]	2014	Control data and control logic	High	Autoregressive model (heuristics) for every variable in the PLC memory	Network: traffic between Supervisory Control and Distributed Control	Centralized	PLC not corrupted; Expet knowledge for the data characterization (classifying system variables)	Process aware attacks	2 real-life water treatment plants	Evaluation of the characterization heuristic (matched process variables)
Y. Hu et al. [80]	2019	Control data and control logic	High	Auto-Regressive Integrated Moving Average (ARIMA) state-space and linear dynamical (LDS)	Not specified	N.A.	Describe entire system with differential equations, i.e. time-invariant system; Expert knowledge of the system for correlation of system variables	Process aware attacks	Matlab simulink	N.A.

V. Havlena et al. [75]	2023	Communication between equipment	Low	DPAs	Network: traffic between Supervisory Control and Distributed Control	N.A.	Representative and sufficient data for the learning phase (attack-free)	Process aware (sequence attacks) – MITRE coverage provided	Datasets of smartgrids (3 days of traffic)	TP, FN
M. Ike et al. [90]	2023	Tasks and resources of equipment + Control data and control logic	High	State-based model where each state is a SCADA phase allowing specific API calls	Host: SCADA via an hypervisor	Centralized	Adapted for OPC-based ICS deployment only ; Windows-based SCADA; Hypervisor is trusted and cannot be compromised	Process aware attacks	Sandia National Laboratory testbed	TP, FP
F. Khorrami et al. [92]	2016	Control data and control logic	High	Modelization of system's dynamic; Differential Equations	Not specified – actuators and sensors' signals	N.A.	Describe entire system with differential equations, i.e. time-invariant system	Process aware attacks, mainly data injection on actuators	HIL	N.A.
O. Koucham et al. [97, 98]	2016 – 2018	Control data and control logic	High	Set of rules expressed with Temporal Logic (LTL, MTL) obtained by data mining – Runtime verification	Network: traffic between Supervisory Control and Distributed Control	Centralized	Adapted for sequential systems only	Process aware attacks	HIL	Mining time for the specifications; monitors overhead, number of properties
A. Kleinmann et al. [94]	2016	Communication between equipment	Low	DFAs	Network: traffic between Supervisory Control and Distributed Control	N.A.	Periodicity and regularity of industrial network traffic	Attack that do not respect communication patterns	Dataset	False alarms
C.-Y. Lin et al. [103, 104]	2018	Communication between equipment	Low	Learning-based approach - pattern mining based on Probabilistic Suffix Tree (PST) to model timing patterns of spontaneous events	Network: traffic between Supervisory Control and Distributed Control	N.A.	Representative and sufficient data for the learning phase (attack free)	Attack containing spontaneous events	Testbed	Length of dataset, average accuracy, prediction accuracy and Kappa value
D. Myers et al. [119]	2017	Task and resources of equipment	Low	Petri net generated by process discovery	Host: use of log files	N.A.	Data describing complete execution of the system; Requires a large volume of data	Attacks manipulating actuators (orders coming from Supervisory control or HMIs)	Testbed (4 Siemens PLC S7-1200)	Time measures
R. Mitchell et al. [114, 115]	2013 - 2015	Control data and control logic	High	Rule-based model; Finite state automata	Host	N.A.	Knowledge about the entire set of critical states	Process aware attacks, mainly data injection on actuators	Simulation	FPR, FNR
K. Miao et al. [112]	2020	Control data and control logic	High	Modelization of system's dynamic; Differential Equations	Network: traffic at local loop level (between a computer and an ARM microprocessor)	Centralized	Continuous-time linear systems; Ability to describe entire system with differential equations (knowledge of system physics)	Process aware especially actuator false data injection	Computer, ARM microprocessor, servo dive and motor (presence of a CAN bus)	FPR, FNR, Recall, Precision, FPR, FNR and F1-score
S. McLaughlin [110, 111]	2013 – 2015	Tasks and resources of equipment + Control data and control logic	High	Rule based model using a state space language (sslang) for online execution and Linear Temporal Logic (LTL) for offline PLC program check ; PLC code instrumentation	Network: traffic between Supervisory Control and Distributed Control and at local loop level	Centralized	Knowledge about the entire set of critical states	False data injection on actuators	Simulation platform (TrymSim)	Policy check times, overhead
J. Nivethan et al. [122]	2016	Control data and control logic	High	Rule-based model; using Zeek	Network: traffic between Supervisory Control and Distributed Control	Centralized	Expert knowledge of systems' variables	Process aware	N.A.	N.A.

G. K. Ndonda et al. [120]	2022	Control data and control logic	High	Temporal rule-based model obtained from learning algorithm (identification of critical process values and extraction of time patterns from normal network traffic)	Network	N.A.	Representative and sufficient data for the learning phase (attack-free)	Process aware	Testbed and simulation	DR, FPR, time to detection
S. Papa et al. [126]	2012	Control data and control logic	High	Transfer function; built with combination of actuators/ sensors measures	Network: traffic between Supervisory Control and Distributed Control	Centralized	Ability to descibe the entire system with transfer functions, i.e. only applicable to time-invariant system	Attacks targeting process data	Matlab Simulation	Time elapsed between the attack and its detection
F. Pasqualetti et al. [127]	2013	Control data and control logic	High	Modelization of system's dynamic; Differential Equations and graph theory	N.A.	Centralized and distributed detection	Continuous-time linear systems; Ability to describe entire system with differential equations (knowledge of system physics)	DoS attacks, stealth, (dynamic) false data injection, replay attacks	Simulation (IEEE 118 bus system)	Detection residual curves
F. Sicard et al. [141, 142]	2018 – 2019	Control data and control logic	High	DFAs and Petri nets; shortest path calculations to calculate distance from forbidden states	Network: traffic at local loop level ideally or data from sensors and actuators	Centralized	Adapted to discrete-event system only; Knowledge about the entire set of critical states.	Process aware; 4 types of attacks defined: direct, sequential, temporal and over soliciting attacks	Simulation of different systems	Complexity of the state space
V. Sharma et al. [139]	2019	Control data and control logic	High	Rule-based model; Set of rules obtained via device's operational profile	Host: embedded IoT equipment	N.A.	Expert knowledge needed for specifying operation profile of embedded IoT device	Process aware attacks	Simulation of a UAV (Unmanned Aerial Vehicle)	FNR, FPR, TPR, Compliance degree
C. Tsigkanos et al. [150]	2021	Control data and control logic	High	Rule-based model using Metric First-Order Temporal Logic (MFOTL); Runtime verification	Network: IoT Cloud Technology	Distributed	Expert knowledge needed to establish the set of specifications	Process aware attacks	Testbed	Event processing throughput, end-to-end latency
D. Urbina [152]	2016	Control data and control logic	High	Modelization of system's dynamic; Auto-Regressive (AR) models or Linear Dynamical State-space (LDS) models	Network: traffic between Supervisory Control and Distributed Control	Centralized	Describe entire system with differential equations, i.e. time-invariant system; Expert knowledge of the system for correlation of system variables	Process aware attacks, mainly data injection on sensors and actuators	Reel system (>100 controllers; Testbed (6 PLCs + 6 backup PLCs)	FPR, TPR, New performance indicators (Stateful vs. stateless, max deviation, etc.)
K. Wolsing et al. [156]	2022	Control data and control logic	High	Learning-based approach proposing 4 independent IDSs: MinMax, Gradient, Steadytime and Histogram.	Network: traffic between Supervisory Control and Distributed Control	N.A.	Representative and sufficient data for the learning phase (attack-free)	Process aware attacks	Dataset	Detected attacks, FP, Accuracy, Precision, Recall, F1-score
M-K. Yoon et al. [157]	2014	Communication between equipment	Low	Bayesian Network + Probabilistic Suffix Tree	Network: traffic between Supervisory Control and Distributed Control	N.A.	Protocol dependant (Modbus/TCP); Periodicity and regularity of industrial network traffic	Sequence attacks (playing with order and timing of commands)	Dataset obtained from a Modbus network testbed	FPR, Receiver Operating Characterisitc (ROC) curve

Table 2: IDS approaches presented in this state of the art

D) Summary and discussion

Note that Table 2 is not exhaustive but aims at representing a global view of the different trends in the literature concerning behavioral intrusion detection. As shown in this table, most of the works combine multiple of the previously introduced approaches: communication between equipment, task and resources of equipment, control data and control logic. Combination of multiple approaches is motivated by the necessity to detect a broader set of attacks.

Chronologically, the first papers mostly focus on the communication between equipment. Approaches relying on task and resources of equipment or control data and control logic started to become more frequent after the years 2010. The approaches focusing on tasks and resources of components quickly appeared to be limited because of their difficulty of deployment due to resource-constrained equipment. And only few works are dedicated to this field of research.

On the other side, detection approaches based on control data and control logic are the main focus in recent years. The necessity of process aware intrusion detection is motivated by recent attacks getting more and more sophisticated and targeting physical processes of ICSs. Therefore, many process aware approaches appeared in recent years. Some hybrid approaches started to raise as well, connecting communication-based detection together with process aware detection.

From this state of the art of behavioral intrusion detection, various observations can be made. All approaches combined, diverse trends exist concerning: implementation techniques for the model construction, data source location, and context aware detection. These aspects are discussed below:

Implementation techniques

Diverse implementation techniques are employed for characterizing a system's behavior, such as time series, DFAs, DPAs, auto-regressive models, modelization of laws of physics, etc. According to the previously introduced taxonomy [64], these implementation techniques can be grouped into three main categories: (i) statistical-based, (ii) (system) knowledge based and (iii) ML-based approaches.

Statistical-based and ML-based approaches do not require *a priori* knowledge about the normal activity of a system; instead, they have the ability to learn the expected behaviors from observations (usually network traffic). This is a good asset but it also leads to inevitable drawbacks. Large attack-free datasets are therefore needed. First of all, proving that a dataset is free of attacks with representative and sufficient data is not straightforward. Secondly, real ICS data for training learning-based model is often missing. And when it is not, the quality of training data is often lacking. Moreover, few learning-based approaches encompass physical process meaning (i.e. process variables) into their model, and when they do they usually need a human expert to tune the model by identifying critical process values. Finally, learning-based approaches generally do not consider operating modes or states of system's components.

In regard with the previously stated characteristics, (system) knowledge based approaches seem promising for ICS intrusion detection. In these approaches, the model is constructed with available system knowledge/data (i.e. it encompasses physical process meaning). On the one side, models may be manually constructed by a human expert ("expert knowledge" in Table 2) which is a time consuming task, and it makes models' updates/modifications difficult. On the other side, recent line of works seek to automate the specification of system's behavior. But automatically obtaining high-quality knowledge is often difficult and time-consuming.

Data source location

Most approaches rely on the observation of network traffic between supervisory control and PLCs. This is explained by the lack of tools allowing traffic observation at fieldbus

level. Among the approaches that claim performing intrusion detection at the local loop level (i.e. below PLC level), few have implemented their methodology on physical ICS testbeds. Most of these approaches use simulations, allowing easy instrumentation of sensors/actuators data [111, 142, 80]. Some approaches use process data to construct their detection model and more specifically information of sensors/actuators, but they retrieve the information on TCP/IP networks [60, 122, 54]. Once again, this is due to the difficulty to instrument the traffic capture at lower levels. Consequently, these methodologies suffer from imprecise data that have crossed several networks and may have been denatured along the way. For instance, the detection will be inefficient if the sensors are targeted or PLCs corrupted. Therefore, methods would usefully complement the state of the art by addressing this issue using instrumented tools to **capture data at fieldbus level**.

Context aware detection

Most of existing work focus on analyzing traffic and/or sensor data in isolation without taking the execution context of the system, into account. To our knowledge, [90] is one among few approaches that encompasses the notion of *context* into their detection methodology. Indeed, the authors differentiate SCADA contexts in two categories: "initialization" and "process-control". The latter comprises two sub-phases: "process-monitoring" and "process-altering". For instance, if an attacker needs to setup and connect its tools to the system, it will propagate "initialization" types of commands into the network. If at that time, the system is in "process-control" phase, this will be reported as an abnormal behavior. The main drawback for this approach remains the fact that it leverages SCADA level information solely. The work in [103] focuses on SCADA spontaneous events and how the traffic changes over time. The authors analyze the traffic regarding its phase transitions which are defined as *periods of time that the distribution of inter-arrival times is stable*. A phase transition happens when the changes of the distribution falls outside a certain range. This is an interesting work that permits to soften the ubiquitous hypothesis of periodicity and regularity of industrial network traffic. But on the other side, it relies on the necessity of having representative and sufficient data for the learning phase which is also a strong and binding assumption. The authors adopted a two-hours learning phase which they admitted was too short for capturing timing patterns of their industrial systems. Moreover, this approach lacks physical process knowledge and is located at SCADA level only. In the same vein, the work [158] does not focus on ICSs but concerns automotive intrusion detection. This time the intrusion detection concerns a communication protocol used at fieldbus level in ICS: CAN protocol. The authors point out the existence of different vehicle driving modes and the fact that approaches relying on constant intervals between messages will fail during transitions of such modes. They propose an approach based on message frequency analysis that they prove remains consistent even when the vehicle operating mode of operation changes. Although different in their methodologies, the three aforementioned approaches share a common characteristic: they are able to tackle context change in a system. They built a model that takes into account different operating modes, whether on a small scale (specific equipment mode at fieldbus level) or big scale (operating modes of SCADA system). Such a notion of context makes it possible to significantly reduce the number of false positives, as unanticipated transitions in operating modes or manual interventions on a system are often detected as attacks. Therefore, this notion of context within systems is primordial. It permits to divide a system into different functioning phases and better address the special characteristics of each. Thus, context-aware detection seems to be a promising direction of research.

Summary

Intrusion detection is a growing field of research and it attracts both the research and industry communities. In this Section, we gave an overview of the current state of the art concerning intrusion detection. We started with general concept of IDS, we then described

in details the properties of behavior-based IDS which is the direction we chose for our work. Indeed behavior-based approaches are the most suitable for ICS-specific case mainly because of their ability to detect unknown attacks. Concerning the data source, we argue that instantiating HIDSs is rarely possible to implement in a real industrial system. Additionally, NIDSs offer a wider scope, therefore our work concerns NIDSs.

From the state of the art of behavioral intrusion detection, we discussed many approaches among three categories: communication between equipment, task and resources of equipment, control data and control logic. This last category is the one that has the highest knowledge of the physical process of a system. We consider that this is an important aspect for intrusion detection against novel sophisticated attacks. Moreover, approaches that take into account the notion of *time*, and characterize the temporal dynamics of ICSs, seem very promising to us. Furthermore, we highlighted the difficulty to collect data at local loop level, and the fact that very few works have put it into practice.

2.2 Runtime Verification

In this Section, we focus on *runtime verification* (also known as *runtime monitoring*) which we identified as a suitable formal verification technique for our intrusion detection approach. Indeed, it is a lightweight verification method that can be deployed during the execution of a running system.

2.2.1 Definition and Concepts

Runtime verification or runtime monitoring is the study of methods to analyze the dynamic behavior of a computational system. The term *verification* implies the notion of correctness, whereas the term *monitoring* refers to the act of observing and evaluating the behavior of a system over time [16]. Runtime verification approaches are based on extracting information from a running system and using it to detect observed behaviours satisfying or violating certain properties. The central object of this approach is called a *monitor* which constitutes the decision procedure for a given property [17]. It verifies violations in system executions by checking the satisfaction of this property. The action of generating a monitor from a property is called the *monitor synthesis*.

To be more precise, in our case, we refer to the properties as "security properties". The monitored system is an ICS and it is instrumented by means of network taps. A network tap (or sensor, or probe) is a network device able to duplicate traffic from a single physical link [147]. Details on the implementation of our approach is provided in Part II.

Therefore, a monitor takes as inputs *execution traces* [136] which are sequences of observations that represent the behavior of interest in the monitored system. These observations can be events or sampled signals indexed or time-stamped. The monitor is capable of analyzing the execution traces (during online execution) and to emit verdicts according to the current satisfaction of a security property. This verdict takes the form of a *truth value* (generally *true* or *false*) from a truth domain that indicates whether or not the run complies with the specification. The overview of the runtime verification process is illustrated in Figure 6. In the context of ICS intrusion detection, we take the assumption that an intrusion targets the physical process leading to violation of at least one security property (which leads to safety violations). Then, intrusion detection is performed by getting verdicts from a set of monitors.

2.2.2 Specification Languages for Runtime Verification

In order to describe the security properties, *specification languages* are used. A specification language is a formal language used to describe a system at a higher level than a programming language. If the specification language is ambiguous (e.g. a natural language

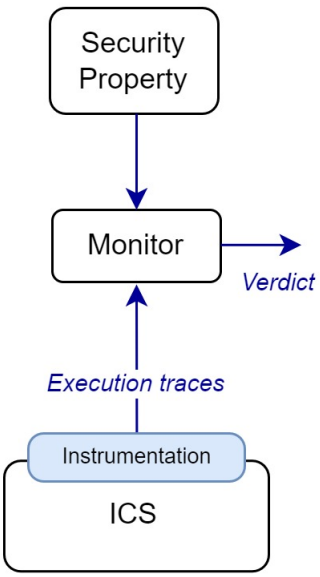

Figure 6: Overview of the runtime verification process applied to an ICS

such as English or French) then the specific property being described may not be clear. Generally, a specification language is expected to be unambiguous and formally structured (i.e. encompassing a syntax and semantics). Some general features are discussed followed by a presentation of Temporal Logic (TL) which is the most common family of specification languages for runtime verification.

A) General specification languages features

Some general concepts concerning such languages are provided in [17] which we are going to rely on to put our work into context.

- **Executable/Declarative.** Some specification languages are executable (e.g. state machines) and others are declarative (e.g. TL). Executable specification languages tend to be more low-level and can therefore be more straightforward to use. On the other hand, declarative specification languages capture properties at a higher level of abstraction. Yet, declarative languages can be translated to executable objects (monitors) through an automated procedure.
- **Finite/Infinite.** Some specification languages are more adapted to specifying sets of finite traces (such as state machines), whereas others are more adapted to specifying sets of infinite traces (such as TL). However, when verifying a system at runtime, the observations are necessarily (ever-increasing) finite traces. This often leads to a mapping from infinite to finite semantics.
- **Time.** There is a distinction between qualitative and quantitative time. Some specification languages can only denote the relative order of states/events on a trace: this is *qualitative* time. Others include a notion of time distance between states/events or range over time intervals: this is *quantitative* time. For instance, specification languages supporting qualitative time can check whether Event A occurred before Event B. Whereas specification languages supporting quantitative time could check if A occurred less than 10 seconds before B.
- **Data.** A specification language may view events as atomic symbols or as structures containing data. Generally, specification languages consider data in various ways, depending on the underlying formalism.

Using runtime verification for ICS intrusion detection guides our choices in the features described above. As we need to monitor security properties, if a declarative language is chosen, there must exist a related procedure to generate executable monitors. Then, a specification language suited for a set of finite traces is required, since in ICS execution traces are (ever increasing) finite traces. Moreover, we need to be able to express both qualitative and quantitative time properties, in order to cover the hybrid dynamics of industrial systems. Finally, the considered events are of various types from logical states, to process variables evolving through time (i.e. functions of time). Once again, this requirement is necessary for capturing the dynamic of ICSs.

B) Temporal Logic

TL is the most common family of specification languages for runtime verification. It is a declarative specification language usually interpreted over infinite traces with adaptions to finite traces. The family includes variants supporting both qualitative and quantitative time. In our work, we focus on three variants of TL: (i) LTL, (ii) MTL and (iii) STL which together cover the variety of temporal behaviors of hybrid systems required for our intrusion detection.

Linear Temporal Logic (LTL)

Let us start with an informal description of LTL. LTL [130] augments *propositional logic* with logical and temporal operators. It permits to express occurrence and ordering in events in qualitative time.

Usual *propositional logic* uses propositional formulas constructed from atomic propositions combined by logical connectives. An Atomic Proposition (AP) is a statement or assertion that must be true or false and that cannot be broken down into smaller sentences. Usual connectives are: not ($\neg$), and ($\wedge$), or ($\vee$), conditional/implies ($\rightarrow$) and biconditional/equivalent ($\equiv$). Hence, propositional logic deals with propositions (which can be true or false) and relations between propositions.

Now, LTL extends propositional logic, with temporal operators:

1. *Always* operator (also known as Globally), denoted $\Box$ (or G). If φ and ψ are propositional formulas evaluated at a position in a trace then $\Box\varphi$ is satisfied (is true) if φ is true at all the subsequent positions.

2. *Eventually* (or Finally), denoted $\Diamond$ (or F). $\Diamond\varphi$ is true if there exists a subsequent position where φ is true.

3. *Next*, denoted $\bigcirc$ (or X). $\bigcirc\varphi$ is satisfied if φ is true at the next position.

4. *Until* denoted $\mathcal{U}$. $\varphi\,\mathcal{U}\psi$ is true if φ is true until ψ becomes true.

Note that in some works, additional temporal operators may be used such as *Release* ($\mathcal{R}$), *Weak until* ($\mathcal{W}$) and *Strong release* ($\mathcal{M}$).

The main temporal operators, previously introduced, are graphically represented in Figure 7. In the diagrams, black dots represent chronological states of the system at various points in time (only qualitative time is described with LTL). These states are labeled with LTL formulas (φ, ψ, etc.). In other words, the diagrams represent infinite sequences; which are of the form $w : \{\psi\}\{\psi, \phi\}\{\phi\}...$. For example, in the first diagram representing the sequence

$$w : \{arbitrary\}\{\phi\}...\{arbitrary\}\{arbitrary\}...,$$

the formula $\bigcirc\varphi$ is satisfied at the first position since φ is true at the next position.

Now, we give the formal definition of LTL's syntax and semantics. In the following, the symbol :: means "is defined as" following [130] notation. $\models$ is the *satisfaction* and $\not\models$ its

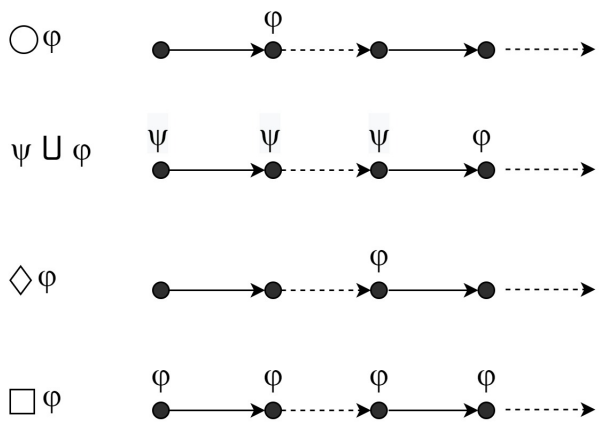

Figure 7: Diagrams of common LTL temporal operators

negation. The symbol | either means "such that (s.t.)" or is a set-builder notation, as in usual propositional logic.

With AP a set of atomic propositions, $\Sigma = 2^{AP}$ a language and LTL formulas $\varphi, \varphi_1, \varphi_2$, the syntax of LTL is defined as:

$$\varphi :: true \mid a \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \,\mathcal{U}\, \varphi_2 \mid \bigcirc\varphi \quad where\ a \in AP.$$

With Σ^ω the set of infinite sequences over the alphabet Σ, $\varphi, \varphi_1, \varphi_2$ LTL formulas, $i \in \mathbb{N}$ a position and $w(i)$ the i^{th} element of the infinite sequence $w \in \Sigma^\omega$, LTL formulas are interpreted using the rules:

$$\begin{aligned}
&w, i \models true \\
&w, i \models a \in AP \quad &&\iff a \in w(i) \\
&w, i \models \neg\varphi \quad &&\iff w, i \not\models \varphi \\
&w, i \models \varphi_1 \wedge \varphi_2 \quad &&\iff w, i \vDash \varphi_1 \wedge w, i \vDash \varphi_2 \\
&w, i \models \varphi_1 \,\mathcal{U}\, \varphi_2 \quad &&\iff \exists k \in \mathbb{N}, k \geq i \quad w, k \models \varphi_2 \wedge \forall j, i \leq j < k \quad w, j \models \varphi_1 \\
&w, i \models \bigcirc\varphi \quad &&\iff w, i+1 \models \varphi
\end{aligned}$$

We also define

$$\begin{aligned}
\Diamond\varphi &\equiv true \,\mathcal{U}\, \varphi \\
\Box\varphi &\equiv \neg\Diamond\neg\varphi
\end{aligned}$$

The remaining operators ($\vee$, $\rightarrow$, $\Diamond$, $\Box$) can be derived from the rules above.

Metric Temporal Logic (MTL)

If the notion of range over time intervals is needed, MTL [18] is more adapted. MTL extends LTL with time measurements over boolean signals. It allows to specify qualitative temporal behaviors by adding the notion of range over time intervals. For instance, the formula $\Box_{[0,3]}a$ asserts that a should hold between 0 and 3 time points from now.

Contrary to LTL, MTL formulas are interpreted over timed sequences which associate every time point with a set of true propositional variables. Timed sequences are of the form $\rho : \{\tau_0, E_0\}\{\tau_1, E_1\}...\{\tau_n, E_n\}$ where $\tau_i \in \mathbb{R}^+$ and is strictly increasing, and E_i is a set of boolean variables which are true at τ_i.

Formally, the syntax of MTL over the alphabet $\Sigma = 2^{AP}$, with $\varphi, \varphi_1, \varphi_2$ MTL formulas, is defined as follows:

$$\varphi :: true \mid a \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \, \mathcal{U}_I \, \varphi_2 \mid \bigcirc_I \varphi \quad where \; a \in AP$$

with $I \subseteq (0, \infty)$ a non-empty interval over $\mathbb{N}$.

Given a timed sequence ρ, MTL formulas $\varphi, \varphi_1, \varphi_2$, interpretation rules are derived from LTL:

$$\begin{aligned}
&\rho, i \models true \\
&\rho, i \models a \in AP \quad &&\iff a \in E_i \\
&\rho, i \models \neg\varphi \quad &&\iff \rho, i \not\models \varphi \\
&\rho, i \models \varphi_1 \wedge \varphi_2 \quad &&\iff \rho, i \models \varphi_1 \wedge \rho, i \models \varphi_2 \\
&\rho, i \models \varphi_1 \mathcal{U}_I \varphi_2 \quad &&\iff \exists k \geq i, \tau_k - \tau_i \in I \quad \wedge \quad \rho, k \models \varphi_2 \quad \wedge \quad \forall j, i \leq j < k \quad \rho, j \models \varphi_1 \\
&\rho, i \models \bigcirc_I \varphi \quad &&\iff \rho, i+1 \models \varphi \quad \wedge \quad \tau_{i+1} - \tau_i \in I
\end{aligned}$$

Signal Temporal Logic (STL)

Signal Temporal Logic [108] is to be used when real-valued (continuous) signals need to be described. STL extends MTL over signal predicates. Signal predicates are used as atomic formulas: $\{x_1(t), \ldots, x_n(t)\}$ together with threshold predicates of the form $x_i \geq 0$ where a signal $x_i(t)$ is a function from a time domain to a value domain. Hence, atomic predicates are of the form: $\mu = f(x_1(t), \ldots, x_n(t)) > 0$.

In the same way as MTL, STL formulas are interpreted over timed real-valued traces of the form $(time, value)$. Therefore, timed sequences are of the form $\rho : \{t_0, S_0\}\{t_1, S_1\}...\{t_n, S_n\}$ where $t_i \in \mathbb{R}^+$ and is strictly increasing, and S_i is a set of signal predicates which are true at t_i.

STL syntax modifies MTL as follows:

$$\varphi :: true \mid \mu \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \, \mathcal{U}_I \, \varphi_2, \quad where \; \mu \in S_i, \mu = f(x_1(t), \ldots, x_n(t)) > 0$$

STL interpretation rules are derived from MTL rules. A supplementary rule is added:

$$\rho, t \models \mu \iff \mu = f(x_1(t), \ldots, x_n(t)) > 0$$

As it is the case for MTL and LTL, the verification of a property allows to determine a binary (true/false) correctness answer. Sometimes this type of answer may not be sufficient, and some quantitative degree of satisfaction/violation would be preferable. To tackle this issue, some authors [51] introduced a *robustness degree* function for STL. This function of time computes the distance to violation of a given STL formula. For instance, if we take the signal predicate $x > c$, the robustness gives the relative position of x to c instead of only indicating whether x is above or below the threshold.

Example of properties expressed through LTL, MTL and STL

Let us consider a motor allowing a part to move on an axis. We are going to express some temporal properties relative to this system. The set of atomic proposition AP might for instance contain propositions about the state of actuators or sensors (such as the motor state or the position of the part on the axis). For instance $Target_{reached}$ and $Motor_{stopped}$ could be the names of two atomic propositions that are respectively true if the target is reached and the motor at standstill, and false otherwise.

Temporal logic formulas can be constructed now with these APs to specify properties of the system. Depending on the need to use qualitative or quantitative time and depending on the nature of the atomic predicate to specify, different TL formalisms may be chosen. The

constraint that the motor has to stop at the next time position whenever the target is reached can be expressed as the LTL formula:

$$\Box(Target_{reached} \rightarrow \bigcirc Motor_{stopped}) \tag{1}$$

The constraint that the motor must run at least once each hour can be expressed as the following MTL formula:

$$\Box(\Diamond_{[0,3600s]} Motor_{run}) \tag{2}$$

The constraint that the motor speed takes between 100 and 200 milliseconds to reach its final value which is bounded between 290 and 310 radians per second can be expressed as the following STL formula:

$$\Diamond_{[100,200ms]}(\Box\ (290 < s(t) < 310)) \tag{3}$$

with $s(t)$ a function of the motor speed $(rad.s^{-1})$ depending on time (ms)

These three properties, numbered from (1) to (3), are graphically depicted in Figure 8.

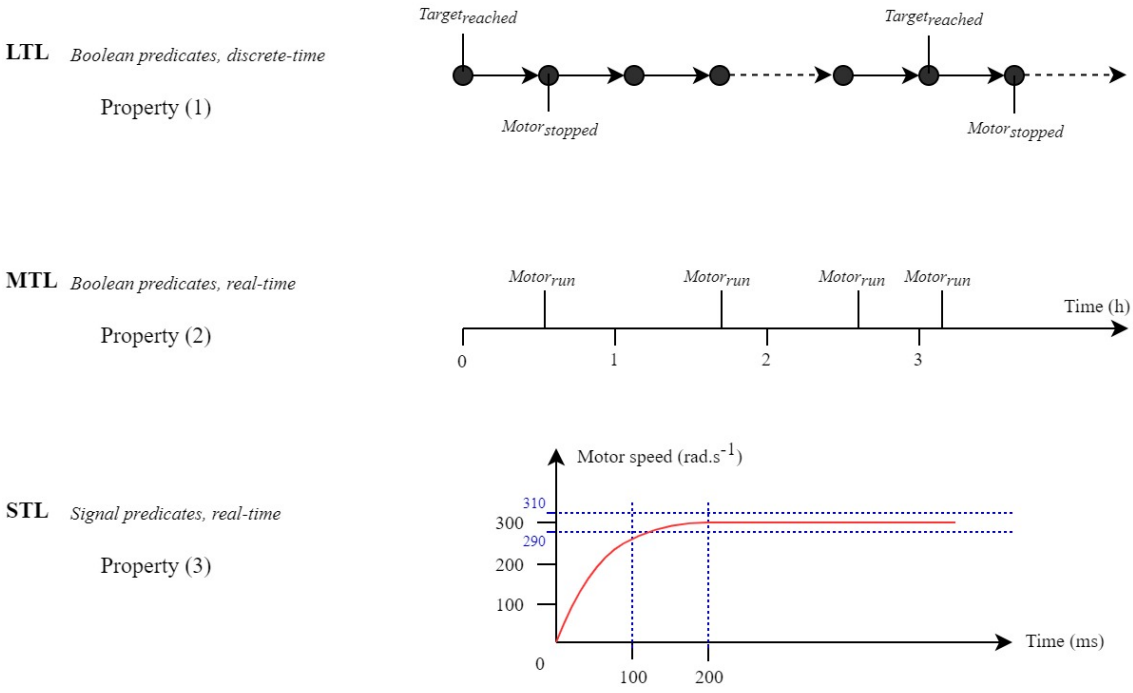

Figure 8: Example of LTL, MTL and STL properties

2.2.3 Monitoring Techniques

Specification formalisms using future temporal operators (Always, Until, Eventually and Next) are typically interpreted over infinite traces. This the case for the formalisms presented above: LTL, MTL and STL. However, the application of these formalisms to the observation of ICS behaviors has to cope with finite (expanding) sequences. Consequently, observing a finite sequence may not be sufficient to determine the satisfaction or the violation of a temporal property according to its formal semantics. Therefore, it is necessary to provide an alternative TL semantics to adapt to finite traces. The problem of interpreting TL over finite or truncated behaviors may be referred as the *finitary interpretation* [16].

A) Finitary interpretation

First of all, we have to mention that some TL variants may be directly interpreted over finite traces, contrary to standard TL. The temporal operators previously introduced are *future operators*: Always, Until, Eventually and Next. Variant formalisms may introduce *Past operators* which are analogous to future ones, but looking into the past: respectively *historically* noted ■, *since* noted $\mathcal{S}$, *once* noted ♦ and *previous* noted ●. The satisfaction of a future operator at position t depends on the values of the sequence w at some or all the positions from t onward. The part of the sequence that is considered is called *the suffix* of w from t to $|w|$. For example $\Box p$ is true at any t if and only if p holds at every $t' \geq t$. The symmetrical past formula $\blacksquare p$ is true at t if and only if p holds at any position $t' \leq t$, in other words, along *the prefix* of w from 0 to t. The motivation to use past operators is that they are straightforward to use and they can provide more natural and convenient specifications [161]. Using only past operators interpreted over finite trace permits to be free from the finitarity interpretation issue. However, past-time Temporal Logic (ptTL) does not yield the expressive power of full temporal logic [49].

Many other works propose to adapt or redefine the semantics of standard TL over finite (and possibly truncated) traces [21, 48].

In [48], the authors introduce *weak* and *strong* views of the finitary semantics for LTL. This work tackles situation when a trace is truncated before the evaluation of a formula was completed, i.e. when there is a doubt regarding what would have been the truth value if the trace had not been truncated. For instance, taking the formula $\Diamond\varphi$ on a truncated path such that φ does not hold for any state or the formula $\Box\varphi$ on a truncated path, such that φ holds for every state: in both cases, we cannot be sure whether or not the formula holds on the original untruncated trace. The *strong* and *weak* interpretation of temporal operators allow to express a verdict for such cases. Therefore in the *weak view*, the formula $\Diamond\varphi$ holds for any finite path while $\Box\varphi$ is satisfied if and only if φ is not violated during the duration of the trace. In the *strong view*, the property $\Diamond\varphi$ is satisfied if and only if φ holds at any time before the trace ends, while $\Box\varphi$ does not hold for any finite path.

Concerning the verdict, a common approach is to use a 2-valued verdict (either true or false), as presented in the above sections. But typically the 2-valued approach does not allow to provide a truth value for every formula in the specific case of finite traces. Indeed, in monitoring a property over finite sequences, three different situations can occur: (i) the property is satisfied after a finite number of steps, independently of the future continuation; (ii) the property is refuted for every possible continuation, and (iii) the finite, already observed prefix still allows different continuations leading to either satisfaction or falsification.

This is why many works proposed a 3-valued verdict domain instead of a 2-valued one. For instance, the authors in [20], propose the following 3-valued semantics: *true*, *false* and *inconclusive*. The three different truth values permit to emit a verdict regarding the satisfaction/violation of a formula when the truncation of the trace occurs. In this work, given a finite sequence u and a formula φ, the truth values are defined as expected: if there is no continuation of u satisfying φ, the verdict returns false. If every continuation of u satisfies φ, the verdict is true. Otherwise, it is inconclusive, since the observations so far do not permit to either return true or false. The authors call their logic LTL_3 and also construct a deterministic FSM with three output colors corresponding to this 3-valued logic[7]. The authors prove that their monitor generation procedure is optimal in two things: first, the size of the generated deterministic monitor is minimal, and second, the monitor is able to execute online monitoring as traces are monitored continuously. The approach permits to identify satisfying or falsifying properties as soon as a decisive event occurs. Hence, a violation is detected *exactly* when it occurs.

[7] https://ltl3tools.sourceforge.net/

However, behind the inconclusive verdict, lies the fact that some properties remain non-monitorable. This means that some formulas can never be satisfied (e.g. $\Box\Diamond\varphi$), i.e. there are always bad extensions.

B) Monitorability

Monitorability refers to the capacity of a monitor, after any finite number of observations, to detect the violation or satisfaction of a property. That is, a monitor should run only if it has the possibility to reach a verdict (with a possible continuation of the observed sequence). To give an example, let us consider the property $\Diamond\varphi$. This property cannot be refuted since φ may appear at any time in the future; and once φ happens, we know that the property is satisfied, independent on any continuation. It means that the property is monitorable. Now, if you consider the property $\Box\Diamond\varphi$, we can never hope to reach a point where it is possible to decide whether this property is violated or not. For whatever happens, we cannot guarantee or refute that φ will still appear an infinite number of times. Thus, the monitor still needs to look up for occurrences of φ in future points in time. This state of indecision of the monitor is caused by a non-monitorable property.

Several works characterize monitorable properties [22, 2]. Namely, the notions of *safety* ($\Box\varphi$) and *co-safety* ($\Diamond\varphi$) properties are introduced. Informally, a safety property states that "something bad should never happen" and a co-safety "something good happens". The union of safety and co-safety properties forms a strict subset of the monitorable properties [21]. However, there remains many non-monitorable properties. The work [21] extends the previously introduced work [20] with the 3-valued semantics [*true, false, inconclusive*]. In [21], a 4-valued semantics is proposed by giving a more detailed evaluation of the *inconclusive* verdict. Informally consider the following example: the property $\Box(a \rightarrow \Diamond b)$ is non-monitorable and is therefore always evaluated to an inconclusive verdict. For such a property, the authors propose the following semantics:

- If the trace ends with the occurrence of b, then the property is evaluated to a truth value that indicates that it is probably satisfied. This verdict is called *presumably true*.
- If the trace ends with the occurrence of a, the property is evaluated to a truth value that indicates that it is likely to remain unsatisfied. This verdict is called *presumably false*.

This new formalism allows to enlarge the set of monitorable properties [22].

In the remainder, we adopt the *strong* and *weak* views [48]. More specifically, we use the *strong* interpretation of the eventually ($\Diamond$) operator: the property $\Diamond\varphi$ is satisfied if and only if φ holds at any time before the trace end. And similarly the *weak* interpretation of the *always* ($\Box$) operator: the property $\Box\varphi$ is satisfied if and only if φ is not violated during the duration of the trace. Furthermore, we only use monitorable properties. Based on the work in [21], we are able to evaluate properties of the form $\Box(a \rightarrow \Diamond b)$. Indeed, in real-life control system, many properties of this type are used, with a being a request/command and b the answer.

C) Implementation of the decision procedure

Two main techniques exist for the implementation of the operational decision procedure when a property is verified by a monitor: *rewriting-based* techniques and *automata-based* techniques.

Rewriting-based technique. A rewriting system does not need any prior processing and operates directly on the sequence of observations in an event-by-event manner [74]. After each processed state, the monitored property is rewritten into a new property that has to

hold in the next state. Then, this newly generated property is evaluated on the remaining sequence. This rewriting process continues until a truth value for the rewritten property is reached.

Consider for example the property expressed earlier as Formula 1: $\square(Target_{reached} \rightarrow \bigcirc Motor_{stopped})$. This property is violated if and only if a $Motor_{stopped}$ event is not observed at the next position after the occurrence of a $Target_{reached}$ event. The monitoring requirement of the monitor (the formula itself) will not change unless a $Target_{reached}$ event happens. In such case, the formula will change to $\bigcirc Motor_{stopped} \wedge \square(Target_{reached} \rightarrow \bigcirc Motor_{stopped})$. A $Motor_{stopped}$ event at the following step will turn it back into the initial formula, whereas any other event will turn it into false. Note that this process is immediate and a violation is detected as soon as it occurs. In practice, evaluating a state predicate and rewriting a formula is a time consuming task. Thus, even though rewriting-based techniques are typically more straightforward to implement, they may suffer from worse runtime performance due to the number of rewrites that have to be processed.

Automata-based technique. Contrary to rewriting-based techniques, automata-based techniques require prior processing before any observations of the system to monitor. During this offline step, monitors are synthesized as FSMs. Constructing a monitor usually requires translating a TL formula into a Büchi automaton [153] which accepts prefixes of infinite sequences. The prefix allows deciding the satisfaction or violation of the given temporal formula. Hence, a finite state automaton permits to recognize as early as possible bad prefixes and reports violation. In Büchi automaton, since the automaton represents infinite sequences (i.e. infinite words in automata theory), the notion of final state does not exist, and is replaced by the notion of *acceptance condition*. Therefore, a run of the automaton is *accepting* if and only if accepting states (often denoted by a double circle) are visited infinitely often. In [153], a formal definition of Büchi automaton applied to LTL is given. For quantitative specification languages, explicit clocks are added and monitors can be synthesized as timed-automata [121].

If we take again the example formula: $\square(Target_{reached} \rightarrow \bigcirc Motor_{stopped})$ with the set of atomic propositions $AP = \{Target_{reached}, Motor_{stopped}\}$, infinite sequences are constructed over subsets of AP and the corresponding automaton represents these sequences. Figure 9 shows a *complete* Büchi automaton for the above LTL formula. Complete, means that any possible sequence over AP is recognized by some run, i.e. from each state a transition exists for each input symbol. In this example, not all runs are accepting: only runs that loop and continuously visit State 0. The small arrow next to State 0 identifies the initial state. A transition labelled 1 should be read as *true*, i.e. the boolean formula accepts any valuation of the atomic propositions.

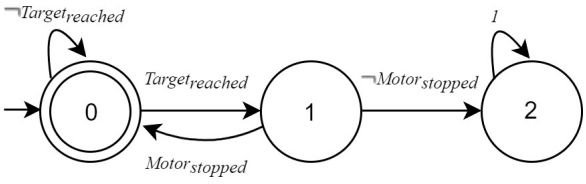

Figure 9: Büchi automaton for the formula $\square(Target_{reached} \rightarrow \bigcirc Motor_{stopped})$

Note that the previous Büchi automaton can be used for the verification of infinite sequences. As already discussed, the typical obstacle in runtime verification for ICS is that only finite traces are manipulated. Therefore, the automaton can be adapted for TL on finite sequences. Hence, the FSM in Figure 10 is generated from the LTL_3 tools already discussed previously, created by the authors in [20] and following their three-valued logic LTL_3. One of their tools automatically generates a monitor under the form of a FSM with 3 possible state categories. Red states are "bad states" (i.e. if a transition leads to that state, the formula was

violated), green are "good states" (i.e. the formula was satisfied), and yellow are "inconclusive states" (i.e. the formula was neither satisfied nor violated). With this model, a transition can possibly be an empty set of propositions taken from AP (noted "Empty" on the Figure). On Figure 10, there are no accepting states. States 0 and 1 are inconclusive and State 2 is a bad state which would immediately trigger an alert when entered during online monitoring.

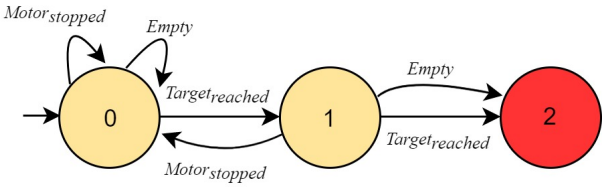

Figure 10: FSM generated by LTL$_3$ Tools for the formula $\square(Target_{reached} \rightarrow \bigcirc Motor_{stopped})$

In our work, we use this tool to generate our monitors, by further refining the inconclusive states, using ideas from the work of the same authors in [21]. Hence, the inconclusive states are divided into "presumably true" and "presumably false" following a 4-valued logic. Details on our implementation are given in Part II.

This technique permits the generation of optimal FSMs statically, i.e. before the monitoring of the system. Hence, this creates a minimal runtime overhead since monitor synthesis is done offline. The only drawback is the memory space required for the offline generation process. However, during the online monitoring process, the time performances are typically better for automata-based techniques. Since runtime performance is critical in the case of ICS intrusion detection, we favor automata-based techniques in our work. Concerning the "reusability of monitor's code" requirement, and for optimization purposes, we also use in our work *specification patterns*.

2.2.4 Specification Patterns

While TL provides suitable formalisms to specify temporal behaviors of system, expressing properties as formulas can be a tedious task. Properties may grow in size and complexity and the task of writing a correct formula appears to be challenging. To overcome this issue, the authors in [47] introduced *specification patterns*. *Specification patterns* are generalized descriptions of requirements on the permissible state/event sequences in a finite-state system. They represent a (finite) set of patterns that occur commonly in the specification of systems. The authors proved that giving a dataset of 555 property specifications (from more than 35 sources and diverse application domains), they could derive almost all of them as their proposed patterns system. These patterns are generic and highly facilitate the specification of critical properties for industrial systems. This is a very good asset for our intrusion detection task, and we can rely on specification patterns to express our security properties.

Several classes of specification patterns were proposed in the literature corresponding to different types of temporal properties. They can be divided into two categories: *qualitative specification patterns* which represent qualitative time properties, and *quantitative specification patterns* which represent quantitative time properties. The distinction between qualitative and quantitative time has already been discussed previously.

In our work, we consider complex processes exposing both time-based and event-based dynamics. In order to describe the variety of temporal behaviors of hybrid systems, we use different categories of patterns. For example, in a system, the properties to be respected can be of different types: (i) sequences of events (e.g. Event A has to follow Event B), (ii) periodicity of sensors reading (iii) or process values that have to stay within a given range.

Each one of the previous examples belongs respectively to a different category of patterns: (i) events order from the qualitative temporal patterns studied by Dwyer *et al.* [47], (ii) time

measurement related patterns from the quantitative patterns proposed by Konrad *et al.* [96] and (iii) continuous signals constraints from the extended quantitative patterns proposed by Maler *et al.* [107].

Patterns are high-level abstractions of temporal specifications and they have to be mapped onto an adequate formalism in order to be evaluated. Dwyer patterns, may be mapped, for instance to Linear Temporal Logic (LTL) formulas. Konrad patterns, are expressed in Metric Temporal Logic (MTL), while Maler patterns rely on Signal Temporal Logic (STL). Each category of patterns is described below.

Dwyer patterns [47]. Dwyer defines two categories of patterns: *Occurrence* of events patterns and relative *Order* of events patterns. *Occurrence* patterns are: *Absence* (an event never occurs), *Universality* (an event is permanently present), *Existence* (an event eventually occurs) and *Bounded Existence* (an event must occur k-times). *Order* patterns are: *Precedence* (Event A must occur before B), *Response* (Event A must be followed by B); with two variations for sequences of events: *Chain Precedence* and *Chain Response*. These patterns are represented in Figure 11.

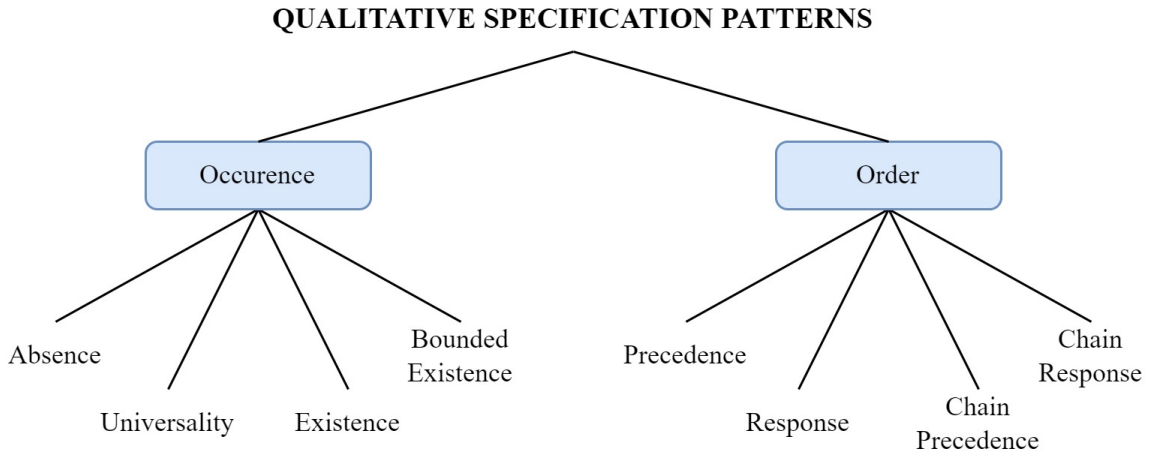

Figure 11: Qualitative temporal patterns introduced by Dwyer

In order to specify the portion of the execution where a specification should hold, the authors introduce the notion of *scope*. All the patterns hold within a *scope*. There are five kinds of scopes: (i) *Global* (always), (ii) *Before* an event, (iii) *After* an event, (iv) *Between* two events, and (v) *After* an event – *Until* another one. In our work, scopes will permit us to express local constraints and bound security properties to certain contexts of execution of the system.

Konrad patterns [96]. Konrad considers time measurements within the patterns. The scopes remain the same as for qualitative patterns. There are three categories of patterns: *Duration*, *Periodic* and *Real-time Order*. In the category of *Duration*, the patterns are: *Minimum Duration* and *Maximum Duration*. The *Periodic* contains only the *Bounded Recurrence* pattern (an event must occur at least a given number of times within a period). Patterns in *Real-time Order* category are: *Bounded Response* (timeout) and *Bounded Invariance* (minimal holding time). These patterns are represented in Figure 12.

Maler patterns [107]. For continuous process variables, Maler proposed a variant of quantitative patterns specifically designed for continuous signals. No explicit classification of patterns exists as it is the case in Dwyer and Konrad studies. However, from a control system theory point of view, some basic safety properties can be straightforwardly expressed in Maler formalism. In that way, in [108], formulas were defined for *Stabilization* (a continuous signal

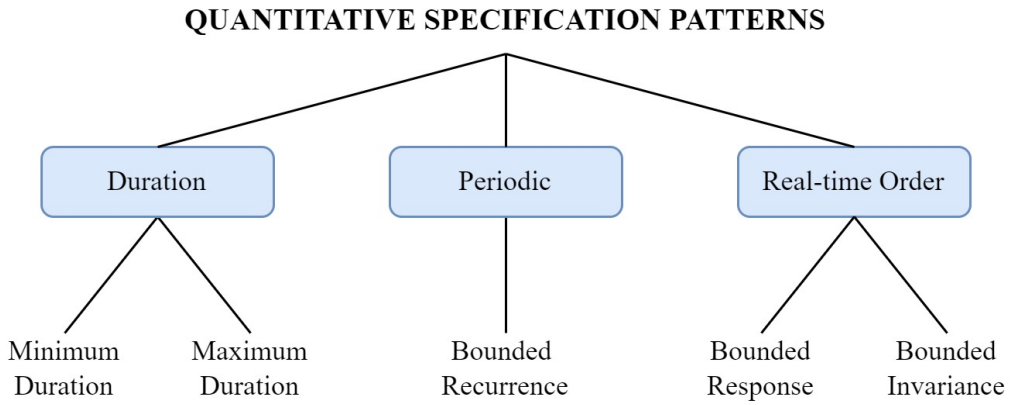

Figure 12: Quantitative temporal patterns introduced by Konrad

value will stay within an interval after a given time), *Bounded* (a continuous signal value will stay inferior to a given limit value) and *Response_ Time* (a signal will reach a given interval within a specified time interval). A generic name for continuous signals properties is not defined yet in the literature. We call these patterns *Maler patterns* as a tribute to Oded Maler, author of the formalism.

Summary

In this Section, we have presented the concept of *runtime verification* which is an effective formal verification technique. If the research community working on runtime verification is active, the related works applied to the field of industrial systems are not abundant. It is a technique that we consider to be very suitable for our work as it enables online monitoring of temporal properties, allowing to capture the complex dynamics of industrial systems. This technique makes use of *monitors* which are capable of analyzing traces during the execution of the observed system. Each monitor verifies the satisfaction of a security property and emits a verdict accordingly, i.e. a *truth value* (either true or false).

We presented three variants of temporal logic – which is the most common family of specification language for runtime verification: LTL, MTL and STL. Using these three formalisms, we are able to cover the variety of temporal behaviors of hybrid systems for the purpose of our intrusion detection task. We then discussed practical aspects of the implementation of runtime monitoring with the notions of the *finitary interpretation* of sequences of events, *monitorability*, and *operational decision procedure* (rewriting-based techniques vs. automata-based ones). Finally, we presented the notion of *specification patterns* that are generalized sets of temporal patterns that could be useful to facilitate system's specification. We identified three pattern systems of interest: Dwyer patterns (qualitative temporal patterns), Konrad patterns (quantitative temporal patterns), and Maler patterns (extended quantitative patterns for continuous signals).

As this Section discussed the theoretical solutions for the online monitoring task, we will discuss in the next Section its possible deployment on large scale ICSs. Such geographically spread out environments call for a distributed intrusion detection approach that could handle local detection alongside global detection.

2.3 Distributed Detection Systems

When discussing practical implementation of an intrusion detection system, the question of its deployment comes up. While the research effort is concentrated on centralized intrusion

detection, we argue that distributed intrusion detection would be more adapted to real-life ICSs presenting geographically spread-out equipment.

2.3.1 Definitions

First of all, we are going to provide definitions of centralized, decentralized and distributed systems. These notions refer to the process by which the activities of an entity are structured in terms of levels of authority and/or decision. These terms are very general and have a variety of meanings depending on the field of application (political science, strategy, planning, control theory, computer networks, etc.). We adopt the communication network view. The terms of *nodes* are used to refer to the processing entity in the different deployments. Depending on each deployment, the nodes have different roles and configurations, detailed below.

We also have to mention the term *Collaborative IDSs (CIDSs)* that appears in the literature, as opposition to isolated or stand-alone IDSs [154, 102]. The authors in [154] give the following definition: *CIDSs consist of several entities* (the nodes) *that act as sensors and collect data. These entities usually contain one or several analysis units carrying out the actual intrusion detection on the data obtained from the sensors.* The concept of CIDSs was introduced to designate sophisticated IDSs frameworks, able to correlate events happening in different places or different networks. A CIDS is therefore able to detect sophisticated attacks involving multiple hosts (sometimes these attacks are called "distributed attacks") on a large part of the network [102]. Depending on their collaboration topology, CIDSs adopt centralized, decentralized or distributed architecture. However, these works do not directly relate to ICS environments. In what follows, we will continue to simply refer to our contribution as an IDS.

The three typical deployments were first introduced in [13] for communication networks and they are represented in Figure 13. Their application to IDSs are detailed below.

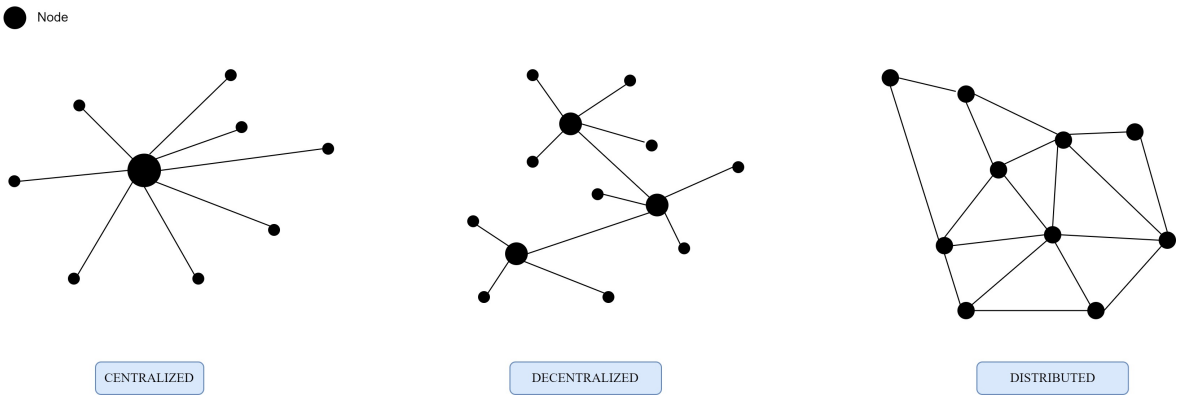

Figure 13: Centralized, decentralized and distributed communication networks

Centralized system. In a centralized system, all nodes are connected to a central processing node. This centralized node takes the advantage of full knowledge of the system [65]. In the case of intrusion detection, the central node is the detection algorithm whereas the other nodes are sensors. Most of the time, sensor nodes are directly forwarding data to the central node which is in charge of storage and processing every data for the detection task. A centralized system is generally easy to set up and can be developed quickly.

Here are the main characteristics of a centralized system:

- *Single intrusion detection entity*: a central node that does all the processing task while every sensor nodes forward the data. This central node is generally a computer running the detection algorithms, to which network taps are directly linked.

- *Global clock*: the synchronization between nodes is easy since the sensor nodes only pass on timestamped data without prior processing. All sensors synchronize with the global clock, i.e. the clock of the central node.
- *Single point of failure*: due to the inherent architecture of centralized communication networks, the whole system is vulnerable to failure. The failure of some sensors would perturb the central node and, reciprocally, a failure of the central node would annihilate the detection task.
- *Difficult scalability*: centralized architecture are difficult to scale up. This is mainly due to the limit in the number of central node connections allowed at a time.

Decentralized system. In a decentralized system, there are multiple processing nodes, each of them making its own decisions. The final behavior of the system is the aggregate of the decisions of the individual nodes. In the case of intrusion detection, there would be multiple detection algorithm nodes, each one linked to sensor nodes. Each detection algorithm node would exercise the detection task locally where it is based, without cooperating with other detection nodes. A major concern on such configuration lies in that the detection performance may be sub-optimal due to individual detection nodes not having as much information regarding the state of the system, as in a centralized configuration [65].

Here are the main characteristics of a decentralized system:

- *Multiple intrusion detection entities*: more than one central entity exercises the processing task. Therefore, connections with sensor nodes can be shared, allowing to divide the intrusion detection task into several strategic points in space.
- *Lack of global clock*: since every node is independent of each other, they have different clocks that they run and follow.
- *Dependent failure of components*: only a part of the system is vulnerable to failure, not the whole system. If some nodes are not operating correctly, the intrusion detection task would still operate on the unaffected global nodes.
- *Improved scalability*: decentralized architecture are more easy to scale up than centralized systems. New intrusion detection algorithm nodes can be added with respective sensor nodes which helps to distribute the workload and increase the detection capacity.

Distributed system. In a distributed system, there are multiple processing nodes sharing resources in a synchronized manner in order to achieve a common objective. In the case of intrusion detection, each node is a detection algorithm node (together with its sensors). The main difference with decentralized intrusion detection lies in the fact that the detection algorithm nodes are synchronized and share useful data. Thus, it allows detection at the vicinity of each detection node independently, but it also enables a more high level detection with aggregated information from every detection nodes.

Here are the main characteristics of a distributed system:

- *Multiple intrusion detection entities*: the intrusion detection task can be geographically spread in space.
- *Synchronization of clocks*: the lack of a common computer clock can cause difficulty in the temporal ordering of events, since each node needs to communicate for the intrusion detection purposes. Hence, a synchronization between nodes is required. This can be performed through Network Time Protocol (NTP)[8] or Precision Time Protocol (PTP)[9] which are networking protocols for clock synchronization over a computer network.

[8] https://datatracker.ietf.org/doc/html/rfc5905
[9] https://standards.ieee.org/ieee/1588/4355/

- *Tolerant to failure*: due to their architecture, distributed systems tend to remove bottlenecks or central points of failure from a system.

- *Good scalability*: distributed systems are highly scalable as they can be easily expanded by adding new nodes to the network. This would allow the intrusion detection system to handle a large amount of data and traffic without compromising on performance, which is particularly useful for large, complex systems.

2.3.2 Motivations

In the context of intrusion detection for industrial systems, our choice is oriented towards a distributed deployment of the intrusion detection task.

First of all, a distributed deployment appears to be particularly adapted to large-scale industrial systems. Indeed, large-scale ICSs are typically geographically spread out and they operate in a distributed manner (see Section 1.1.2). It appears to be more intuitive to perform intrusion detection in a decentralized manner onto a system that is itself controlled in a distributed way. Indeed, it allows to monitor a larger portion of the network (compared to isolated detection) since the processing tasks are shared. Hence, the respective sets of observable events are not the same for every detection algorithm node. In other words, due to the distributed nature of the system, the different detection algorithms process data from different local loops allowing to divide the detection task, by processing locally the effect of different sets of sensors and actuators. Furthermore, processing events locally allows to propagate only crucial information generally resulting from aggregation of local data or local verdicts while avoiding the propagation of useless information on potentially several kilometers. Therefore, a more high level detection based on such aggregated events (such as the termination state of a machine for example) is made possible with a synchronization of each detection algorithm node.

From a performance point of view, distributed deployment permits to distribute the computational load required for the detection task and keep processing and memory overhead as low as possible without impacting the monitoring capabilities. Advantages of a distributed configuration include modularity, scalability and robustness [65].

Another motivation for using a distributed deployment for intrusion detection is the ability to detect wider categories of attacks. In a distributed intrusion detection system, knowledge about current local states can be aggregated and exchanged in a cooperative manner. Hence, multiple local information may be required for the monitoring of a more high level security property. This allows to detect global system violation that would not have been detectable from any local view solely. This point is demonstrated in [56], where the authors present a "one-to-many" attack scenario, i.e. a single attacker launching attacks against several components. It is shown that a single intrusion detection node is not aware of the entire attack scenario since its knowledge is limited to local events. Therefore, detecting this scenario requires collaboration between concerned detection nodes.

2.3.3 Challenges of Distributed Deployments

A) Communication

One of the main challenges in distributed system deployment is the *communication* between nodes, since node cooperation heavily relies on the communication network medium that connects spatially distributed entities. The authors in [65] state that this issue is often the main factor to cause system performance degradation. As defined by the authors in their work, the *communication* issue may include (but are not limited to): network-induced delays, data packet dropouts, data packet disorder, quantization error, time-varying network

topology, network channel fading and time-varying network throughput. In our work, we face primarily:

Network-induced delays. We face more specifically *transmission delays* in the communication network which are dependant on network conditions such as network channel quality and network traffic congestion due to a limited bandwidth of a communication network. Such constraints in the communication resources are to be asserted during the design and implementation of the distributed intrusion detection system. Indeed communication networks may possess limited bandwidth and restricted transmission rates. For instance, Ethernet communication network features are defined in the IEEE 802 family of standards[10].

Data packet disorder. This issue occurs when timestamped data transmitted by some nodes arrives at the destination node in a different temporal order. This situation usually occurs because of network-induced delays, and especially when the delay is greater than the prescribed sampling period of the received events.

B) Runtime Verification

Another challenge concerns the practical implementation of monitoring a global system requirement, i.e. a requirement that includes several local nodes information. In the context of runtime verification, these requirements translate as properties, that we will call *global properties* in the sequel (in opposition to local properties).

Due to the distributed nature of industrial systems, monitoring a *global property* requires to gather data from many distinct traces from different local views of the system. In the case of a distributed deployment for the intrusion detection task, no (virtual) global trace of the system is directly available, as it would be the case in a centralized/decentralized deployment. Indeed, in the latter, a global trace could easily be reconstructed from local traces to simplify the evaluation of global properties. In distributed deployments, by nature, there are no central data collection point where all the component's behavior would be observable. Therefore, the question of monitoring such global properties need to be addressed.

Let us provide an example of such a global property. Take a system S composed of two subsystems $S = \{A, B\}$. The subsystems A and B are industrial tanks containing liquids. These tanks are equipped with sensors that give indication on the liquid level (for tank A, $a0$ means low and $a1$ means high whereas for tank B, $b0$ means low and $b1$ means high). For each subsystem we define the set of atomic propositions: $AP_A = \{a0, a1\}$ and $AP_B = \{b0, b1\}$. These two subsystems are monitored locally in an autonomous manner. Local intrusion detection nodes are able to monitor local security properties including local predicates. However, monitoring a global security property is not straightforward since it includes predicates from both subsystems. For example, one requirement of the system S could be that "Tank A and B must not be empty at the same time". This is an *Absence* pattern that can be written in LTL as the following property $\varphi = \Box(\neg(a0 \wedge b0))$. Each tank has information on its own liquid level but has no way of knowing the level of the other. Thus, this is a global security property. This example is illustrated in Figure 14.

The challenge is therefore to define an efficient methodology to monitor such properties. Generally, either (i) a global point of observation is introduced, in which case the monitoring falls back into the classical TL monitoring problem, or (ii) no global point of observations is introduced. In that case, monitors have to communicate with each other and the monitoring process can take several steps before reaching a verdict. In [19], the authors propose an approach using the rewriting-based technique whereas a similar work is done for automata-based techniques in [118].

To answer the same challenge of monitoring global properties, the authors in [34] propose three possible settings:

- *Orchestration*, where a single monitor conducts all the monitoring process while receiving

[10] https://ieee802.org/

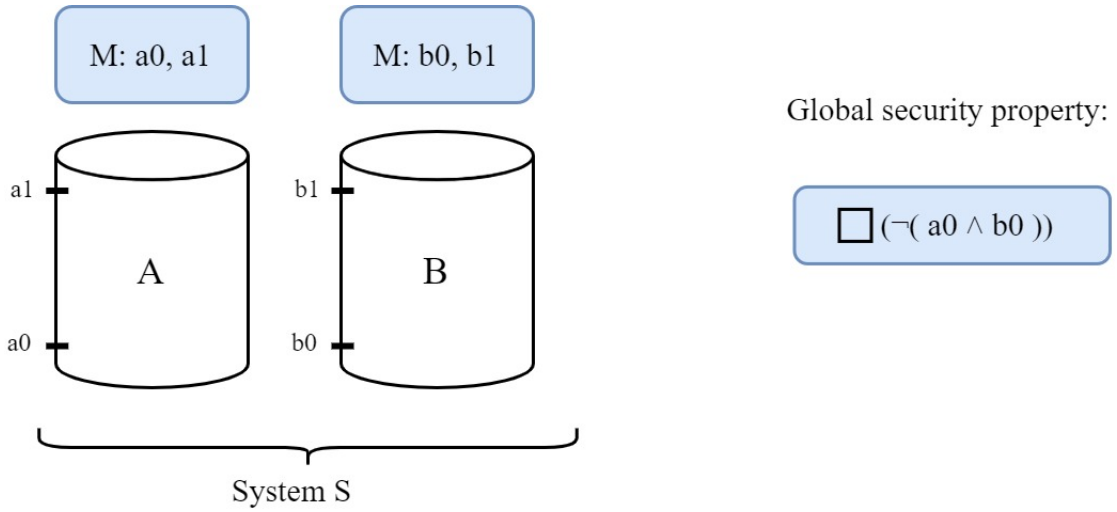

Figure 14: An example of a system's global security property

local events from other local monitors. Note that the processing monitor may either be one of the monitors attached to local components or an additional monitor introduced into the system.

- *Migration*, where the monitoring entity transports itself across the network, gathering required information through the process.
- *Choreography*, where monitors are organized into a network, each of them in charge of monitoring a "projection" of the formula.

Our contribution adopts a setting close to *Orchestration*, in order to address the special case of global security properties. An adaptation is proposed to match both the distributed IDS deployment and the ICS use case. Details are provided in next Part.

2.3.4 Related Work on Distributed Intrusion Detection

In the literature, most of the works deploy a centralized intrusion detection system. This is mostly explained by the straightforward implementation of such deployment and the fact that it is amply sufficient for testbed demonstrations. In such approaches, only locally available data is monitored. In the majority of distributed IDS from the literature, only the data collection task is distributed and not the intrusion detection task itself. In such approaches, the data may be pre-processed locally, and it is in any case forwarded to a more central entity for the purpose of intrusion detection. For instance, in the prior work in [145], the authors presented one of the earliest distributed intrusion detection framework for heterogeneous networked systems. The prototype implements preliminary event analysis at local nodes and a unique centralized analysis, called the DIDS director (where DIDS stands for Distributed Intrusion Detection System). The novelty of the presented architecture was to propose a distributed data collection and pre-processing, performed through individual hosts and Local Area Network (LAN) monitors. However, the main intrusion detection task is centralized.

Similarly, in [117] the authors deploy a rule-based distributed IDS for large-scale smart grids. The authors introduce a master node connected to numerous sensors. Each sensor collects local data in order to forward it to the master node, with no local prepossessing. In [12], the authors propose a hierarchical architecture with two layers: (i) a basic processing layer for input data based on statistical methods and (ii) a central evaluation module based on partially ordered events through Petri nets. The authors prove that the distributed architecture provides better processing balance and accelerates intrusion detection. In this approach, the workload is distributed due to the basic processing on each local node, but

there is still a central module doing the main processing. Even though these methodologies may allow to distribute basic processing tasks and permit to handle heavy network traffic, we argue that they still present a centralized detection structure. The various processing nodes are executing different tasks and they are dependant on each other. Such a deployment is prone to bottlenecks and have single points of failure.

However, few works present a genuine distributed intrusion detection deployment using several cooperating detection nodes. The work in [56] presents a framework for distributed intrusion detection applied to computer networks only. The framework is composed of several collaborating intrusion agents and it operates in a completely distributed manner, i.e., data acquisition, data analysis, and communication tasks are decentralized. Therefore each intrusion agent has a layered structure comprising the following modules: sensor, detection (misuse-based and anomaly-based), alert correlation and reaction. The authors use communication and synchronisation methods from the research field of intelligent agents, but few details are provided on these topics. Furthermore, the method has not been tested on a physical system and relies on a simulation. The authors in [7] address the case of power grid SCADA systems and adopt a distributed methodology that they deem more adapted to the great amount of data generated at each substation. The authors present a tool detecting malformed packets in a wide range of SCADA protocols. The tool connects to network tap interfaces. The approach is distributed as most of the data analysis is performed locally at substations and only aggregated operations are performed at the control center. Hence, the control center performs data aggregation and correlation. However, the paper does not indicate whether local substations have autonomous intrusion detection capacity and if alerts can be raised locally on local events. In [137], the authors propose a distributed intrusion detection framework, composed of multiple IDSs (called the "SCADA intrusion detectors (SCIDs)") placed at every level of the ICS. The authors do not rely on any central components. To our knowledge, it is the only work mentioning field level by placing SCIDs on fieldbuses. Each SCID is tailored to analyze the specific network traffic at its surroundings and is able to detect anomalies. Furthermore, it has the ability to propagate the results of its analysis to other SCIDs in order to identify distributed attacks. The interaction between SCIDs is done through the "publish/subscribe" paradigm. However, the paper is lacking an experimental application and no results are provided.

Summary

In this Section, we have presented three typical network communication deployments for intrusion detection systems: centralized, decentralized and distributed. We focused on distributed deployment for intrusion detection systems as it appears to be particularly adapted to large-scale geographically spread out industrial systems. Indeed, distributed deployments allow to tackle heterogeneous networks with heavy workloads. It also permits to avoid relying on a central detection entity and prevents from single point of failures.

Concerning implementation details, few (if any) distributed approaches from the literature, validated a prototype comprising fieldbus data capture. As previously stated, this is an aspect we tackle in our contribution. Finally, implementing runtime verification in the context of distributed deployments faces some challenges. Local properties relying on a single local execution trace are straightforward to evaluate. But global properties relying on multiple execution traces across the network can be more challenging to evaluate. In the literature, several approaches exist for the monitoring of such properties.

2.4 Conclusion and Positioning

In this Chapter, we presented a taxonomy and a literature overview of intrusion detection methodologies for ICSs. We also presented the concept of runtime verification, a lightweight

verification technique we adopt for our detection task. We finally discussed typical network communication deployments for intrusion detection, with a particular interest towards distributed deployments and its relevance in industrial systems.

The state of the art on IDSs for ICSs shows a great interest in behavior-based approaches. Such approaches are interesting for industrial systems since they allow to detect novel attacks. However, they also present some remaining difficulties. From the literature, we have identified the following weaknesses:

- **Limited comprehension of the physical process.** Behavior-based IDS approaches confronted to the rising trend of ICS tailored attacks favor the detection of process aware attacks. However, many works focus solely on the communication aspects of ICS networks, with a limited comprehension of the physical process. We argue that such approaches lack efficiency against process aware attacks. With respect to such attacks, accurate and efficient intrusion detection requires observations and understanding at the level of the physical process. We also note that physical process knowledge includes different levels of abstraction: (i) Control data and control logic, but also (ii) functioning modes and operational contexts. Furthermore, from a technical aspect, the closer to the physical process is the observation, the better. Industrial fieldbuses observations show very little implementation in the literature, but would be fruitful to implement.

- **Cost of detection model construction.** With respect to specification-based approaches, that we deem promising for behavior-based detection of process aware attacks, the main drawback is the manual characterization of the detection models. Not only this is a time-consuming specification task, but it also requires a deep and exact knowledge of the whole physical process. To reduce the cost of designing the models, a systematical approach for the specification extractions would be more appropriate.

- **Limited model expressiveness.** Many exploratory research works suffer from insufficiently expressive models. For instance, few works encompass explicitly the notion of *time* into their model. The evolution of real-life ICSs through time presents both event-driven and time-driven dynamics that require different formalisms to be studied. In order to capture the full dynamics of industrial systems, adopting a more expressive formalism is required.

- **Limited observation range.** The research effort is concentrated on centralized intrusion detection. Such approaches often generally develop a stand-alone IDS with a limited observation range (data collected in its immediate vicinity) which gives small visibility on the system they intend to protect. Even if such approaches are well adapted to testbed experimentation, they generally suffer from no scalability to spread-out environments. Consequently, they are not able to detect distributed attacks. Very few distributed approaches stand out in the literature. We argue that this direction has to be explored as it would permit to gain observation range and increase the number of detected attacks.

Positioning

In this manuscript, we aim at answering some of the weaknesses mentioned above. Our work consists of a specification-based detection framework directed towards the detection of process aware attacks. We address the main drawback of specification-based approaches (manually derived rules) by presenting a systematic methodology for the specifications extraction from international and industry standards. Moreover, in addition to being systematic, our approach, once deployed, requires only few updates (compared to most specification-based approaches) as they mostly depend on changes in standard specifications. Another originality of our approach lies in the fact that data collection is distributed across the system and includes fieldbus level data collection in addition to Ethernet TCP/IP networks. Our framework

tackles the monitoring of hybrid dynamic systems by making use of expressive formalism from the TL family of specification languages. This allows us to characterize the systems' behavior as security properties over temporal constraints. Our model encompasses physical process behaviors, with both knowledge about control data/logic and context knowledge. Context knowledge concerns operating modes of components or more general functioning phases of the system.

The methodology to design such intrusion detection models is the object of Chapter 4. The deployment of our methodology is detailed step by step in Chapter 5. This instance of our IDS enables detection of security property violations occurring at its vicinity. The detection of a broader category of attacks together with a more scalable framework calls for multiple instances of our IDS framework. Therefore, Chapter 6 details and answers the challenge of a distributed deployment of our methodology. We perform a distributed detection task including the verification of local security properties and the verification of global system security properties. Figure 15 and 16 give a global overview of the positioning of our contributions.

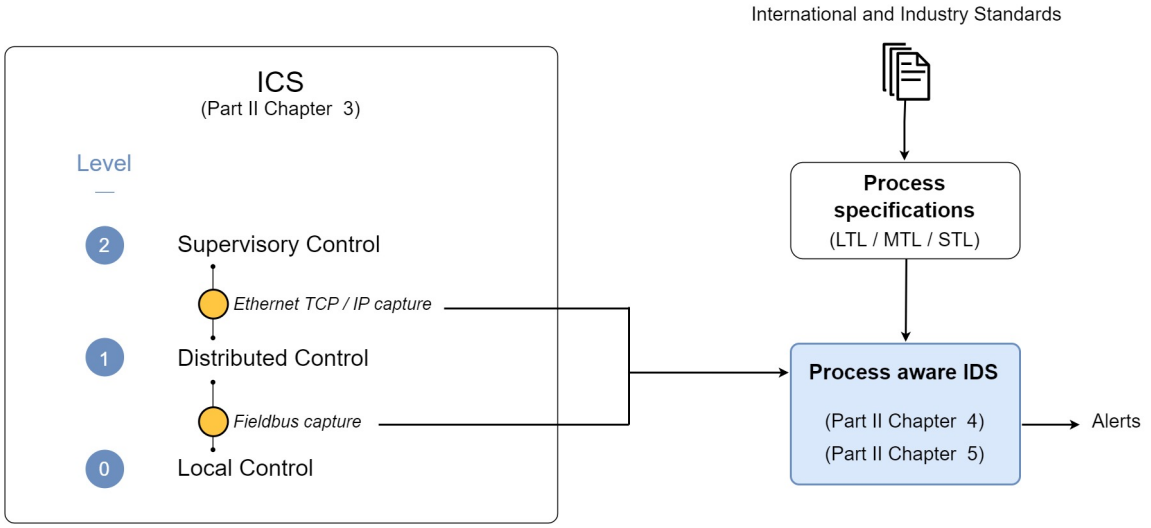

Figure 15: Global positioning of our contributions - Chapters 4 and 5

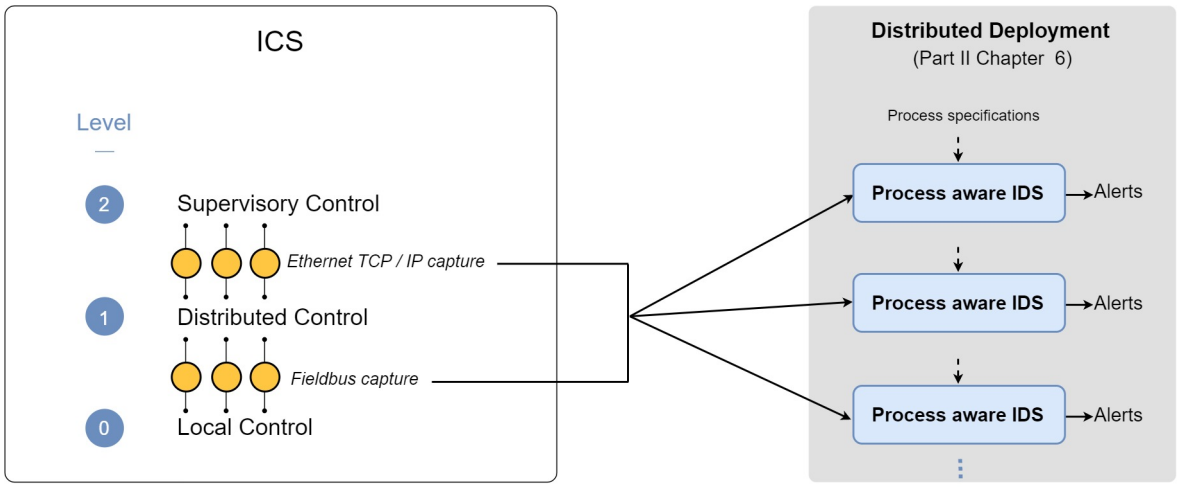

Figure 16: Global positioning of our contributions - Chapter 6

Following the same taxonomy as the one used to review the state of the art of behavioral IDSs for ICSs (see Table 2), our contribution – which aims at answering the 4 previously identified weaknesses – is synthesized in Table 3.

Authors	Year	Approach	Phys. Proc. Knowl.	Implem. Technique	Data Source	Deploy.	Hypothesis	Att. Typo.	Expe.	Evaluation Methodology
E. Hotellier et al. [79]	2024	Control data and control logic	High	Set of rules expressed with TL (LTL, MTL, STL) obtained by safety properties extraction from standards – runtime verification	Network: traffic between Supervisory Control and Distributed Control and at local loop level	Centralized and Distributed	Systematical methodology but not automatic	Process aware attacks	Two physical ICS testbeds	Detection capabilities, Scalability and Extensibility

Table 3: Global characteristics of our contributions

Part II

CONTRIBUTIONS

Chapter 3

Industrial Physical Testbed

Contents

Introduction

Before describing our intrusion detection framework, we want to point out the fact that the experimental part of our work was conducted on two ICSs testbeds.

Cybersecurity research helps to anticipate future attacks. The validation of new countermeasures has to be led on realistic platforms for getting precise and fruitful feedback for cyberdefense. This is the reason why we dedicate this Chapter to an introduction on industrial physical testbeds.

We will start by briefly presenting our motivation for using testbeds, then we give an overview of existing work on industrial physical testbeds. Then, we describe the two ICS physical testbeds we used in our work. Finally, practical details are discussed for capturing network traffic on these testbeds.

3.1 Motivation

The use of industrial platforms offers many benefits to organizations. They provide a realistic and safe testing environment. They are proven to be the most effective tool for awareness and training [9]. Moreover, they can be used for threat detection evaluation. However, the asset we are most interested in here is the functional validation of intrusion detection solutions such as IDSs. The main challenge in industrial systems cybersecurity research is to create a realistic environment. Obviously, it is not possible to execute attacks – with potential disastrous consequences, on a real facility. On the other hand, reproducing a physical process of realistic size is very costly and challenging if not impossible in the case of a dangerous process (e.g., chemical factory). The present Section is dedicated to industrial platform initiatives. We will give several definitions, present the state of the art of ICS physical testbeds with a special focus on virtualized physical processes and maritime testbeds since

these are features of the two testbeds used in our work. Finally, the testbeds used in our experiments are presented.

3.2 Definition and Criteria

With the development of the cybersecurity research on industrial systems, more and more ICS testbeds have been developed for testing intensively the efficiency of cyberdefense measures. Related to this topic, the paper [35] deserves particular attention since it is a recent and complete survey on ICS testbeds for cybersecurity research. The authors define *testbed* as a testing infrastructure, consisting of a scaled-down version of a real ICS, created *ad hoc* to reproduce real-world systems but in a controlled environment. The authors in [35] classify three possible types of testbeds: *physical*, *virtual* and *hybrid* ICS platforms. ICS *physical* testbeds use real hardware and software to configure both the network and physical layers. These testbeds are the most accurate since they are the nearest to the reality. Real experimental conditions (noise, latencies, etc.) lead to a precise and detailed understanding of the system, which will allow more efficient countermeasures (e.g., low False Positive Rate). Moreover, this kind of platform allows to exploit the vulnerabilities of a specific device or protocol implementation weakness. On the contrary, *virtual* ICS testbeds use software simulations, and/or virtual machines, and/or emulations to replicate a system (networks and components). Even if they are more flexible and cheaper solutions than ICS physical testbeds, this option generates data with lower fidelity. Indeed, virtual platforms cannot reproduce the behavior of real field devices like PLCs or drives for instance. Therefore, it is not possible to reproduce threats that exploit vulnerabilities of real devices. On the side of *hybrid* testbeds, they are a conjunction point of the other two categories. They use both physical devices (such as PLCs, HMIs, etc.) and software simulations (of actuators and sensors for example). This solution allows to keep good fidelity due to the physical equipment deployed, but at the same time limits the cost and development time. They represent a good trade-off between *physical* and *virtual* testbeds.

Testbeds must then be differentiated from two other close concepts: Cyber Ranges (CRs) and Digital Twins (DTs). CRs are mainly virtual testbeds dedicated to train operational teams to improve their responsive capacity in case of a cyber crisis, and show safely the consequences of an attack to non-experts (e.g., C-level managers). Clearly, the objectives of testbeds and CRs are different; in testbeds, cybersecurity experts want to experiment countermeasures on a representative version of their system. On the other side, cybersecurity analysts, operators and possibly non-experts use CRs to quickly instantiate exercises of cyberattacks on a given system (possibly not the one the trainees daily operate), to increase or assess their cyber awareness level, or their operational experience. A DT can be defined as a digital representation of a physical object, asset or system (here, either a factory or a ship). The traditional roles of DTs are predictive maintenance, failure analysis and recovery, and alternative architecture design testing. Since Cyber DTs are mainly virtual platforms, they also come with additional challenges [78], and are costly to create, maintain and run. With regard to these definitions, the two platforms used in our work are testbeds, and they are described in Section 3.4.

According to [35], testbeds can be benchmarked and compared through four main criteria: (i) Fidelity, (ii) Repeatability, (iii) Measurement Accuracy, and (iv) Safe execution. Any testbed is a trade-off between all these criteria with respect to various constraints.

(i) Fidelity. A testbed with a high degree of fidelity perfectly replicates the devices and processes from a real-world ICS. The more layers of physical, IT, and OT networks are implemented in a testbed, the more realistic it is.

(ii) Repeatability. Repeatability of the experiments generally means higher confidence. It allows third-parties reproducing attacks and countermeasures under the same conditions

as originally defined, in order to compare findings and solutions on the same system.

(iii) Measurement Accuracy. In order to get accurate and reliable data from an ICS physical testbed, synchronized sensors can be integrated in different places. These passive probes do not disrupt the underlying physical process.

(iv) Safe execution. According to [35], it is difficult (if not impossible) to deploy attacks in a real environment since they can damage the physical process or some devices. Therefore, a testbed can be viewed as a "sandboxed" environment with safe executions of attacks.

(v) Deployment Cost. One could add this fifth criteria, since deploying a testbed can be a very costly operation (especially for physical testbeds).

3.3 ICS Physical Testbeds

The work synthesized in [35] constitutes a recent and complete survey about ICS testbeds and datasets for security research. We do not consider the fully virtual testbeds since they differ too much from the testbeds used in our work. On the other side, fully physical testbeds are most of the time real size processes, built especially for cybersecurity studies. Lots of them are costly nationwide projects, the best known being the Idaho National Laboratory [43, 24] which is a massive industrial site. This testbed encompasses different facilities across multi-disciplinary areas such as an electric power grid, a water treatment system, nuclear facilities, etc. Compared to many physical testbeds, this one is not a scaled-down version of a real system, it is a full-scale site spanning over 2300 km^2. However, it is quite difficult for students or researchers to have access to such an industrial site. Eventually, there was an intense activity in construction of physical testbeds recently [4, 3, 70, 109, 30]. Yet, changing the physical process in these solutions is costly. Therefore, virtualizing the physical process appear to be a good trade-off solution.

Let us consider the platforms on which only the physical process is virtualized since it is the technology of one of the testbed used in our work. Some approaches rely on commercial interface boards in order to connect the physical process and control devices (Arduino shields, Raspberry Pi modules or specialized data acquisition modules) [76, 66]. These interface boards are most of the time limited to digital Input/Output (I/O) and rarely analog I/O. When available, analog outputs are one or two in number which gives very few opportunities. Beyond the communication aspect, few details are given about the process simulation in these approaches.

Many approaches rely on commercial simulation software [61, 99, 3] like Matlab/Simulink[1], OPAL-RT[2], LabVIEW[3], Power-World[4]. Several works use software process simulation modeling. These software are dedicated to a certain kind of process (e.g., chemical industry) [25, 143, 8] and they are mainly used for productivity forecasting. They are not designed to communicate with other components such as PLCs and thus do not have any network interface (such as Modbus). Other approaches are sparsely configurable and are not meant to reproduce an industrial process. Rather, they are used to test and stress SCADA by populating data [36, 44, 45]. Additionally, new generation 3D simulation softwares, like Home I/O[5] or Factory I/O[6] are being developed for technical and professional education. Home I/O is a virtual house with a lot of automated components, whereas Factory I/O allows the user to build a virtual factory by assembling subsystems. Both softwares can be interfaced with real PLCs. Since they are not open source, they remain costly to use for research and/or student

[1] https://www.mathworks.com/products/simulink
[2] https://www.opal-rt.com
[3] https://www.ni.com/labview
[4] https://www.powerworld.com
[5] https://realgames.co/home-io/
[6] https://factoryio.com

projects [128].

Furthermore, special attention is given to maritime testbeds since one of the testbeds used to deploy our work is a replica of a warship. Among related work dedicated to the maritime domain, the authors in [148] proposes a roadmap for the creation of a specialized maritime-cyber lab, which combines maritime technology and traditional cybersecurity labs. The proposed Cyber-SHIP Lab (Software, Hardware, Information and Protections) hosts a range of real, non-simulated, maritime systems. This paper mainly describes the planned future works on Cyber-SHIP platform, e.g. gathering real maritime equipment that can be found on a ship's bridge for pentesting, providing signal simulation to test spoofing and jamming, and software tools for vulnerability and risk analysis. Even if it is impossible to recreate every available bridge setup, the authors in [148] claims that a number of bridge configurations can be achieved with a single lab. The authors admit that the main drawback of Cyber-SHIP Lab is its prohibitive cost. Moreover, many technical details are lacking in this paper, e.g., the detailed architecture of the Lab, its IT/OT components and the considered cyberattacks. It is thus difficult to compare fairly our warship testbed with this reference.

The continuation of this work is described in [149], where the desired platform depicted in [148] is still not completed at the publishing time. The authors in [149] provide a conceptual and generic analysis on how CRs can be used in the maritime context. This article presents a hybrid approach combining real physical data and simulation, where up to 3 physical testbeds can be interconnected. The architecture of their CR is simplified into 3 layers: OT part tends to exist on the physical layers (e.g., propulsion, energy, etc.), while IT part lies on the higher layers (e.g., network packets, operational data). Even interesting, only a few technical details on the CR are provided, for example it is not easy to differentiate what is really physical from virtual, and the implemented cyberattacks and detection means are not described.

The work in [134] proposes a virtual port logistics and supply chain cybersecurity training platform. This CR is built to organize small cyber exercises on specific topics. It is hosted on a rack server, the network side is virtualized, and the computers used in the exercises are running in Virtual Machines (VMs). It enables the simulation of cyberattacks and defences, identification of threats and vulnerabilities, traffic monitoring and in-depth analysis. It consists mainly of customized workflows for setting up the scenarios with virtualized systems. Participants use a remote desktop or similar equipment to connect to the environment. Technical vs. non-technical information can be displayed during the training sessions, depending on trainees' level of technical skills. This testbed is mainly virtual. OT components can be optionally included in their CR, while the testbeds we use in our work are mainly dedicated to this kind of devices. Moreover, the trainees have only visibility (and then, can only act) on user scenarios, and not on network topology, virtual and hardware environment.

The authors in [132] present a maritime CR environment, combining navigational, information and telecommunications systems, networks, and SCADA systems. It supports maritime vulnerability, penetration and exploitation scenarios, traffic eavesdropping, positional systems spoofing, navigation takeover, signal intelligence scenarios, and others. It combines simulated and emulated systems to be as realistic as possible. It is made of two main entities: a single ship or a fleet, and an ashore maritime center. Among all the implemented maritime CR components, the "Machinery Control System" is the most related to our testbeds, since it involves PLCs to get information about the ship's machines characteristics (e.g., temperature, engine speed, fuel supply, etc.). The work presented in [132] is particularly focused on radar and Automatic Identification System (AIS) attacks. Finally, it is claimed that various hacking positions are possible, but no precise attack scenario is given, so it is difficult to both compare with [132] and assess the feasibility of their attacks.

The work presented in [23] is about a testbed for cyber security awareness in the maritime domain. Two subsystems are implemented: the propulsion system with the engine control, and the navigation system with a rudder. Some sensors and actuators (e.g., temperature

sensing) are simulated with Arduinos. The testbed instantiates Supervisory control with a SCADA component implemented with a dedicated HMI. IDSs are also deployed to check some basic safety properties. Three attacks are described: forging a TCP packet with a specific payload to stop the program loop of one automaton, using specific messages to write data to the memory of one automaton, combining two situations that are considered valid individually, but where their conjunction leads to an unfortunate situation. It appears that it is a small scale ICS testbed, with a few integrated components, and only two implemented subsystem loops of a warship.

3.4 Testbeds used in our Work

The first testbed we used for the experimental part our work, is GreEn-ER[7] Industrial Control systems Sandbox (G-ICS) [133]. The second is a representative model of a warship situated in Naval Cyber Laboratory, Naval Group Ollioules, which was presented in our paper [140]. These testbeds were used for the experimental validation of our work as they permit practical attacks and related countermeasures to be safely benchmarked. Let us describe their features.

3.4.1 G-ICS Testbed

G-ICS is a flexible testbed used for both research and training. It is based on a Hardware-In-the-Loop (HIL) technology which combines the advantages of virtualized and physical testbed. The solution adopted in G-ICS virtualizes only the physical process alongside sensors and actuators. These virtualized elements are connected to real industrial control devices (like PLCs, HMIs) using an open source electronic interface system. Figure 17 gives an overview of the Hardware-In-the-Loop solution adopted in G-ICS testbed.

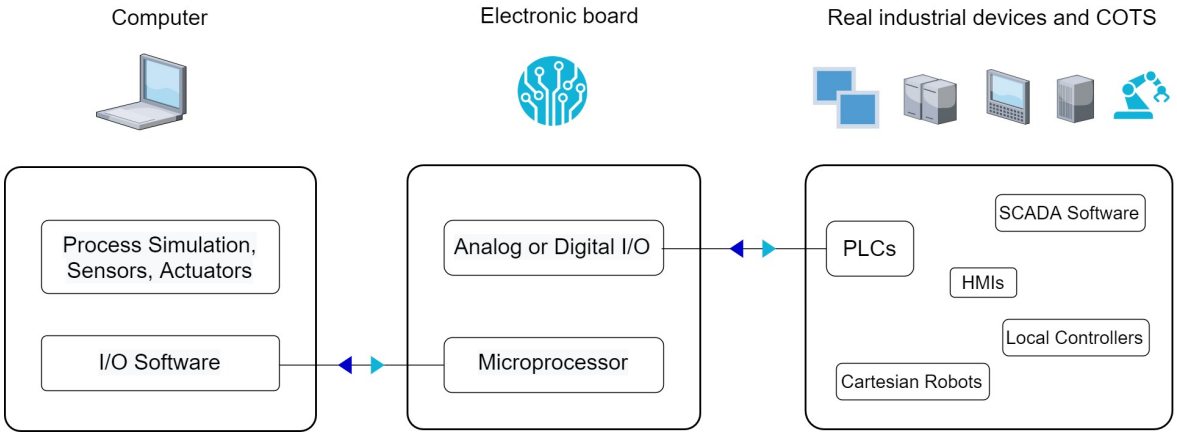

Figure 17: Overview of the HIL system of G-ICS testbed

All different layers of the Purdue Model are represented through a diversity of real industrial devices and commercial SCADA softwares. Added to the physical components, there is a simulation of the physical process, sensors and actuators. Because industrial interfaces from PLCs are not compatible with those of a computer, electronic boards are needed to interface industrial devices and the computer running the simulation. The simulation is a computer program. Between the electronic boards and the simulation, there is a software that takes the information from the virtualized sensors and actuators and passes it on to the analog and/or digital inputs of the electronic boards. The outputs of the board are then directly wired to the analog and/or digital inputs of a PLC or other industrial devices with I/O modules

[7] Grenoble Energie - Enseignement et Recherche

and interfaces. Boards can also integrate industrial serial interfaces such as Modbus Remote Terminal Unit and CAN.

Figure 18: Overview of G-ICS testbed © All rights reserved

In terms of surface area, the testbed takes up a whole room, see Figure 18. Concerning the industrial equipment forming the testbed, there are Commercial-Off-The-Shelf (COTS) (controllers, protection relays, remote terminal units, industrial HMIs, etc.) from several vendors and several supervisory control servers. Added to that, three industrial cartesian robots are also in the room. The robots are called "cartesian" in reference to the *Cartesian coordinate system*. In geometry, this coordinate system specifies uniquely points in space. Typically in two dimensions (which is the case for our robots), each point is positioned in space using pairs of numbers (x axis and a y axis). Here, the three robots are drawing robots using a two axis cartesian positioning system. Thanks to the Hardware-In-the-Loop system, the testbed is highly configurable since any physical process can be simulated and interconnected with the physical components. The first experimental part of our work (Chapter 5) is based on one of the three cartesian robots, further detailed whereas the second part of our work (Chapter 6) uses two cartesian robots in addition to a virtualized manufacturing plant.

3.4.2 Naval Testbed

The naval ICS testbed used in our work implements a representative model of a surface warship. It is composed of a physical part and a device control part, see Figure 19. This environment is representative of a warship in terms of manufacturing brands of the equipment (Siemens and Schneider Electric) that compose it and in the general architecture but it is not similar in terms of complexity and quantity of equipment. The simplified architecture scheme of the testbed is available in Figure 20. For simplicity, some equipment are not represented (for example, industrial firewalls connected to switches).

The architecture implemented in our testbed shows four main functions of the ship: Direction, Energy, Artillery and Propulsion. We call *subsystem loop* the system comprising the local network of a specific maritime function. Four subsystem loops can be identified:

- Direction subsystem (Schneider Electric PLC M340-20): allowing to control the direction of the ship. This involves acting on the rudders position in order to vary the ship direction.

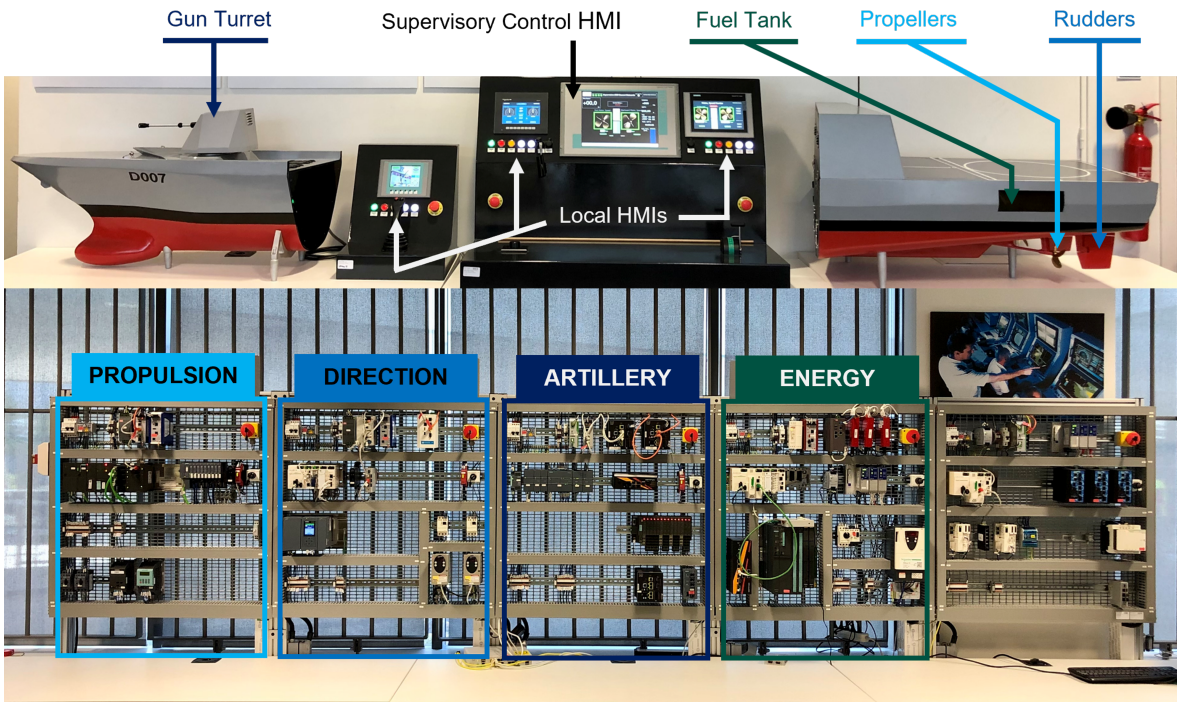

Figure 19: Overview of the warship – HMIs and Physical view (top) and Industrial control devices view (bottom) © Naval Group SA in Sicard Franck, Hotellier Estelle, and Francq Julien. An Industrial Control System Physical Testbed for Naval Defense Cybersecurity Research. In : *IEEE European Symposium on Security and Privacy Workshops (Euro S & P W)*. IEEE, 2022. p. 413-422.

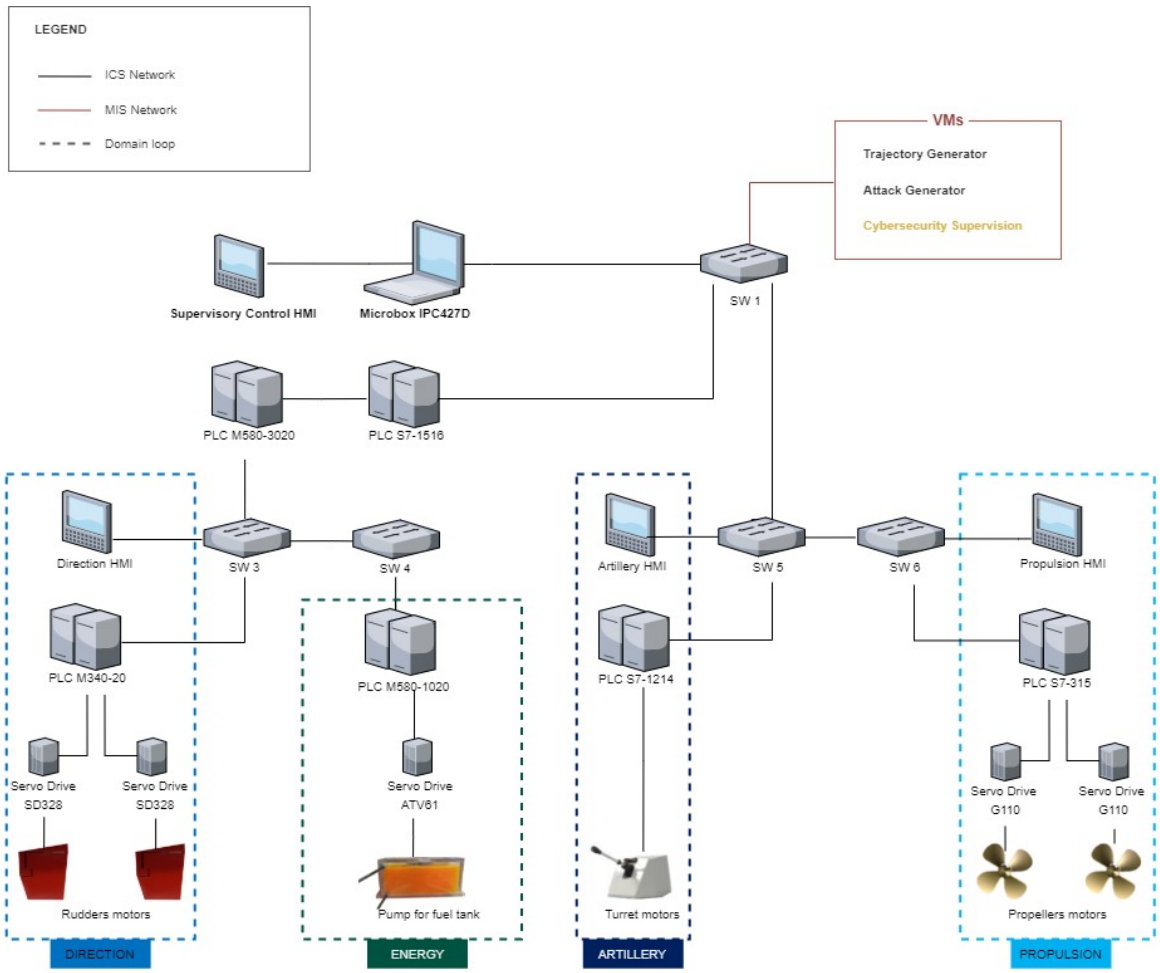

Figure 20: Architecture of the warship

- Energy subsystem (Schneider Electric PLC M580-1020): allowing to supply fuel. It is a functional servitude that operates filling and emptying the fuel tank.
- Artillery subsystem (Siemens PLC S7-1214): allowing to control the 76mm gun turret model. It acts on the positioning of the gun turret (azimuth, elevation) and firing.
- Propulsion subsystem (Siemens PLC S7-315): allowing to control the propulsion of the ship. The aim is to act on the propulsion propellers to vary the speed of progress of the ship.

Each subsystem loop is driven by a PLC. The different PLCs communicate with each other via coordination PLCs (M580-3020 and S7-1516). These coordination PLCs are also used as data hubs for the Supervisory Control. Each subsystem loop is controlled by authorized human operators, either manually on the equipment, locally from a local HMI or remotely from the Supervisory Control HMI. It is important to note that these OT systems (subsystem loops and Supervisory Control) are complemented by an IT level corresponding to Levels 3 to 5 of the Purdue Model (denoted MIS Network in Figure 20). This IT level, is composed of VMs either generating traffic on our ICS physical testbed ("normal" traffic with the Trajectory Generator VM and "malicious" traffic with the Attack Generator VM) or managing cybersecurity information (Cybersecurity Supervision VM).

Concerning the communication between low-level components, there are fieldbus networks using Modbus Remote Terminal Unit and Profibus, whereas higher level components implement Modbus Transmission Control Protocol (TCP) communication protocol.

3.5 Monitoring Network Traffic

From a practical aspect, capturing network traffic is required in order to implement our intrusion detection approach on a testbed. Concerning the implementation of our solution, we decided to rely on an existing open source NIDS since this type of solution is usually supported by a large community. Consequently, it is expected to be maintained and tested in different environments, making it safer than a home-made one. In what follows, we give a rationale for using Zeek in our work compared to some other existing open source IDS.

The considered open source NIDSs are Snort[8], Suricata[9], and Zeek[10].

Community

These three solutions are free and open source. They are supported by a large and active user community. The contributors can be industrials as well as academics.

Detection rules

Every IDS solution relies on detection rules to monitor the network traffic. The three solutions allow users to deploy custom rules.

Snort and Suricata even propose sets of pre-established rules, some available for free and others that can be purchased. Users can easily create new rules and their integration is quite simple and easy to implement. On the contrary, implementing rules with Zeek is not as easy, but offers more flexibility.

General framework

Suricata and Snort are comparable in their framework design. They encompass a detection engine that execute the rules set by the user on the incoming network traffic. They are more

[8] https://www.snort.org/
[9] https://suricata.io/
[10] https://zeek.org/

oriented towards signature analysis. On the contrary, Zeek is more of an anomaly detection system. Its architecture allows users to develop complex rules utilizing scripts to analyze traffic. The scripting module enables temporal analysis between different frames, which is not supported by signature-oriented IDSs.

OSI layers

Most of the IDSs aim to protect companies networks. Therefore, their implementation makes them able to track protocols over TCP or UDP which are Layer 4 of the Open Systems Interconnection (OSI) model. They also tackle protocols over IP, which is Layer 3 of the OSI model. That is the case of the three different IDSs. They also have visibility at the application layer (Layer 7 of the OSI model) and they can identify HTTP or SSH traffic for example. Nevertheless, we have to find a way to run the IDS on Layer 2 of the OSI model since we are going to encompass fieldbuses in our project. On this matter, it is necessary to create a parser for the services we are going to work with.

Adding a parser

From what has been discussed in the previous paragraph, implementing a new parser is a feature that needs to be taken into account for our project. Few information is available about developing a parser for either Suricata or Snort but since the tools are implemented in C, it should be feasible to come up with a new parser. Zeek ensures the maintenance and distribution of BinPAC which is a declarative language and compiler designed to create new parsers [125]. A lot of documents and tutorials are available for using and implementing BinPAC[11] with Zeek.

User experience

Suricata provides a dashboard and is known to be really user friendly. On the contrary, it is said that Snort graphical interface is not very user friendly but the community has worked on this issue and provided many GUIs (Graphical User Interfaces). Concerning Zeek, there is no interface and picking up with the tool can be quite challenging.

Performance

According to the literature, for simple rules and on Linux environment, Suricata shows a better processing rate than the two others [6, 72].

The aforementioned characteristics are summed up in Table 4.

Eventually, Zeek seems to be the most suitable open source IDS for our approach. The main motivations being the following:

- Zeek is more oriented towards anomaly detection while Suricata and Snort master signature-based detection;
- Zeek is highly configurable and offers the possibility to implement sophisticated rules in its own scripting language;
- Adding a parser is a well-documented process that will allow us to encompass Level 2 of the OSI model in our contribution.

[11] https://github.com/zeek/binpac

	Snort	Suricata	Zeek
Community	Active user community		
Detection rules	Pre-established rules can be purchased, Easy to implement	Pre-established rules can be purchased, Easy to implement	More challenging to implement, but offers more flexibility
General framework	More oriented towards signature analysis	More oriented towards signature analysis	More suitable for anomaly detection
OSI layers	From 3 to 7		
Adding a parser	Few information; implementation in C language which is quite popular	Few information; implementation in C language which is quite popular	Well documented; declarative language maintained and distributed by Zeek, specifically designed to create new parsers
User experience	Graphical interface Some projects are implemented by the community to make it more user friendly	Dashboard Really user friendly	No interface Difficult to pick up with
Performance	Lesser performance than Suricata for simple rules	Better performance than the two others for simple rules	Allows sophisticated rules

Table 4: Comparison of open source NIDSs

Summary

In this Chapter, we explained the notion of testbed by giving its definition and criterias it responds to. Testbeds differentiate from CRs and DTs. Testbeds can be physical, virtual or hybrid. Our work is deployed on two physical ICS testbeds that were presented. We also investigated the state of the art concerning testbeds. We noticed that only a few notable effort had been made to implement testbeds in the maritime context. In our opinion, the warship testbed used in our work appears to be one of the most advanced of the maritime state of the art and has a high level of flexibility and repeatability. All sectors combined, G-ICS physical testbed is one of the largest testbeds in terms of number of physical components, with a size close to real industrial cases (over one hundred industrial equipment and over one thousand simulated sensors and actuators). G-ICS has the advantage of being open source and proposes customizable industrial processes with the use of the Hardware-In-the-Loop technology.

Finally, using testbeds will allow us to implement our intrusion detection framework, evaluate our contribution and validate it in safe environments that constitute complete and realistic ICSs. Testbeds allow precise and intensive study of the effects of propagating attacks and corresponding detection means. From a practical aspect we investigated three open source NIDSs and we explained why we chose Zeek in our work.

The research work (i.e. detection framework), presented in Chapter 4, has been deployed on both testbeds. Similar results were obtained and they are discussed in Chapter 5. Aside from this discussion, the implementations details are given for one testbed solely, for clarity and confidentiality matters. Therefore, every implementation examples in Chapter 4, Chapter 5 and Chapter 6 refer to G-ICS testbed.

Chapter 4

Detection Framework

Contents

Introduction

In Part 1, we have defined context on ICSs and we have provided an overview of the state of the art of process-oriented intrusion detection approaches. We have identified weaknesses in the literature and this present Part aims at answering some of them in our contribution.

Therefore, our contribution is a framework for ICS intrusion detection. The aim of this Chapter is to unfold the methodology to be followed to obtain a behavior-based intrusion detection approach which: (i) has a deep knowledge of the physical process and operates at fieldbus level (thus, leveraging the monitoring of the state of actuators/sensors and component's internal states); (ii) presents a systematical specification-based model construction, from international and industry standards (thus reducing the cost of detection model construction); (iii) relies on expressive formalisms that covers the hybrid dynamics of ICSs and (iv) presents a wide observation range due to a distributed detection deployment at different local loops of the system, thus covering a greater diversity of attacks (e.g. distributed attacks).

Before describing in details our framework, we first define our threat model. Secondly, we give a comprehensive overview of our approach. Then, we set out the model construction with the security properties extraction followed by the security patterns synthesis. We finally expose the runtime monitoring process.

4.1 Threat Model

As mentioned in Section 1.3, a threat model defines which parts of the system are targeted, what is the goal of the attacker and which classes of attacks are in the scope of our intrusion detection approach.

First of all, we consider the whole ICS as a potential target. Attackers can target any components from Level 0 to level 2 (see Figure 2 page 10) or any network link between components. In our study, we consider that the threat has already gained access to the system and has control over one or several high-level controllers. Whatever the target is, we aim at detecting effects of an attack on the system i.e. we look for the system's incorrect behaviors. In other words, we leverage the detection of security property violations provoked by attacks. Note that one limitation of our contribution is that we have visibility only on what occurs in the network traffic and some components' internal variables may not be visible in the network traffic. However, a change in internal variables is likely to induce changes in the system's behavior, with visible effects into network traffic.

Secondly, we focus on attackers aiming at disrupting the physical process of an ICS. In other words we focus on *process aware attacks*. We previously introduced process aware attacks as *attacks having a high level of physical process knowledge* (see Part I Section 2.1.3). We can go further and define these attacks as *attacks disrupting the physical process and inducing incorrect behaviors of the system*. Process aware attacks do not violate the syntax of the communication protocol, i.e. the network frames forged by attackers respect protocol specifications in terms of address fields or lengths, for example. Instead, these attacks concern the manipulation of data related to the physical system in order to disrupt it in a stealthy way. Some examples of process aware attacks are: commands using legitimate orders sent out of their context (e.g., stopping a movement before a target is reached), forcing out-of-range values for process variables (e.g., sending a speed setting point out of safe range), inverting valid commands (e.g., inverting the order of products in a tank filling, resulting in sabotaging the recipe), targeting control logic (e.g., forcing transitions or changing internal states of controllers), etc.

Poisoned Water and INDUSTROYER.V2 (discussed in Part I Section 1.2) are examples of real process aware attacks targeting respectively the amount of chemicals in a city water supply and circuit breakers in electric grids. Modifying a control setpoint or opening circuit breakers, can be both legitimate commands that happen in the absence of an attack. However, by forcing process variables into unsafe ranges in the first case, and modifying the temporal ordering of commands in the second, the attacker induced incorrect behaviors of the system. We aim at detecting such attacks whose objective is to disrupt the physical process of ICSs.

Thirdly, we define the threat characteristics (i.e. the class of attacks) with respect to the MITRE ATT&CK for ICS framework [116] (introduced in Part I Section 1.2). This latter provides the relevant tactics for our studies: "Execution", "Collection", "Command & Control", "Inhibit Response Function", "Impair Process Control" and "Impact". The other tactics of the MITRE Matrix are not relevant for our work since they are mostly concerning initial access and deployment of the threat. Our assumption is that the threat is already inside the system. With respect to the considered MITRE tactics, the characteristics of our threat model are the following :

- "Execution";
  The threat is able to change operating modes of controllers and modify controller tasking.

- "Collection";
  The threat is able to perform Man-in-the-Middle (MITM) attacks, detect operating modes, sniff network traffic, monitor the process, get access to the controller program and process variables images. Therefore, all the "techniques" of this MITRE tactic apply, although we do not specifically detect them in our approach. That being said, our methodology is able to detect security property violations provoked by such attacks.

- "Command and Control";

The threat is able to use standard application layer protocols. Again, our approach only detects side-effects of such attacks.

- "Inhibit Response Function";
  All of the techniques listed for this tactic may apply. As we focus on process aware attacks, we do not specifically detect network events corresponding to firmware manipulation. Once again, our detection is able to detect side-effects of such attacks.

- "Impair Process Control";
  All the techniques apply. We assume that messages coming from sensors can be trusted. However, we do handle sensor false data injection attacks in the case of broadcasting networks. This special case is addressed in Section 5.2.

- "Impact";
  We focus on loss of availability, control, protection, safety and view on the process. We also consider damage to property, denial of control, denial of view, manipulation of control and manipulation of view.

This techniques coverage is synthesized in Table 5.

Execution (TA0104)	**Collection** (TA0100)	**Command and Control** (TA0101)	**Inhibit Response Function** (TA0107)	**Impair Process Control** (TA0106)	**Impact** (TA0105)
Change Operating Mode, Modify Controller Tasking.	Adversary-in-the-Middle, Automated Collection, Data from Information Repositories, Detect Operating Mode, I/O Image, Monitor Process State, Point & Tag Identification, Program Upload, Screen Capture, Wireless Sniffing.	Standard Application Layer Protocol.	Activate Firmware Update Mode, Alarm Suppression, Block Command Message, Block Reporting Message, Block Serial COM, Data Destruction, Denial of Service, Device Restart/Shutdown, Manipulate I/O Image, Modify Alarm Settings, Rootkit, Service Stop, System Firmware.	Brute Force I/O, Modify Parameter, Module Firmware, Spoof Reporting Message, Unauthorized Command Message.	Damage to Property, Denial of Control, Denial of View, Loss of Availability, Loss of Control, Loss of Protection, Loss of Safety, Loss of View, Manipulation of Control, Manipulation of View.

Table 5: Threat Model in correspondence with the MITRE ATT&CK ICS Framework

Attacks using the aforementioned techniques are the focus of our work and would be detected by our detection approach (or in certain cases, their effect on the physical process). For a better overview, the threat model's scope of action is identified within the global ICS MITRE Matrix alongside the techniques directly detected by our approach, in Figure 21.

However, as already mentioned in Part I Section 1.2, it is important to note that relying on the MITRE framework can present some limitations. A technique coverage is rarely complete as a technique can be broken down into several expressions. And assuring a technique total coverage assumes knowing the exact number of its expressions which is an information not provided by the framework.

As a conclusion, our threat model is directed towards the entire ICS. The attackers uses process aware attacks which means that they aim at damaging the physical process of the system. In this kind of attacks, the commands used by attackers cannot be easily differentiated from legitimate commands. Most of the attack detection methods inherited from the IT world would not be appropriate in that context. Therefore, a different detection approach is required.

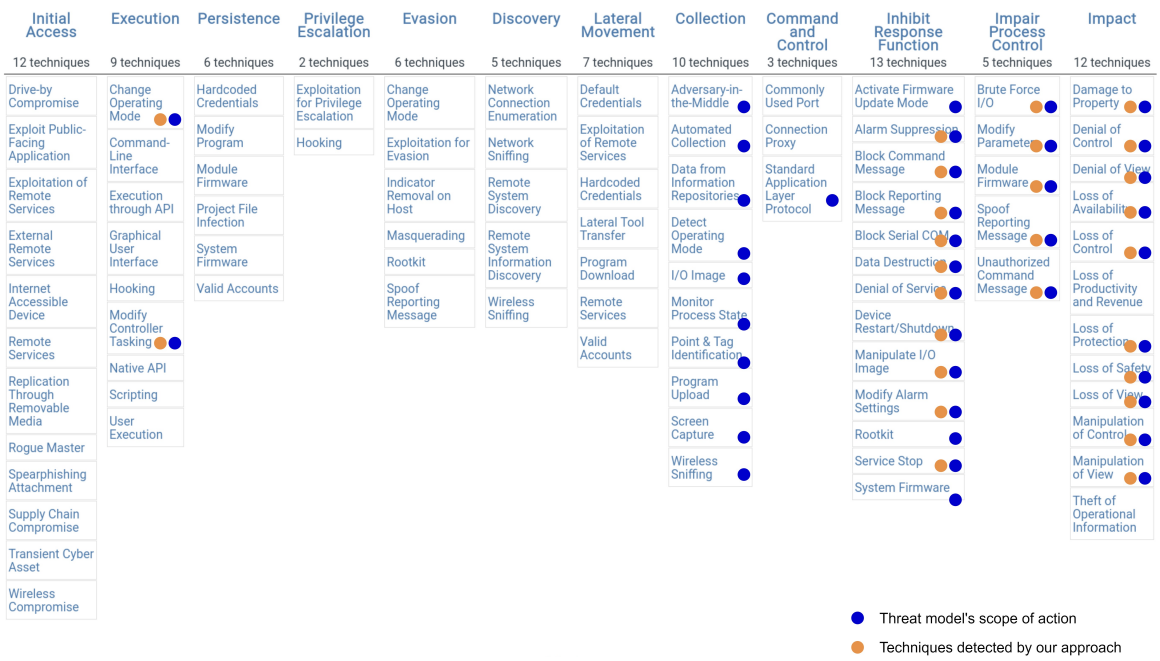

Figure 21: Threat model's scope of action and detection capabilities of our approach *© 2024 The MITRE Corporation. This work is reproduced and distributed with the permission of The MITRE Corporation.*

4.2 Overview of our Detection Approach

A) Context

Here, we want to clarify some notions about the behavior of a dynamical system and the process variables that can be observed in order to construct the IDS model.

Industrial systems are dynamical systems comprising state variables, either continuous or discrete that evolve through time. Informally, a *trajectory* (or *sample path*) is a possible sequence of system states and events over time. Therefore, the behavior of an industrial system corresponds to trajectories in the state space. The *state space* of a system, is the set of all possible values that the state may take. Depending on the application or the considered variables, the state space can be real numbers (subsets of $\mathbb{R}$), but also just values from a discrete set, such as $\{S1,\ S2,\ S3\}$ or $\{Open,\ Closed\}$. Yet, dynamical systems can be classified based on the nature of the state space selected for the model.

Similarly, dynamical systems can also be modeled in continuous or in discrete time. Time is often assumed to be a continuous variable valued over a dense domain (e.g. $\mathbb{R}$) since it corresponds to our basic notion of time in the physical world. But it is often convenient to define the variables of a system at discrete instants only. This is due to the control components forming the systems that are mainly operating in discrete time (discrete-time clock). Therefore, the process is only monitored at specific moments in time (sampling or sporadic observations). As a conclusion, state variables can either be described over continuous or discrete domains, as well as time.

Figure 22 shows the distinction between a continuous-time and discrete-time representation of the same continuous signal $x(t)$. On the left part of the Figure, the signal is represented in continuous-time. On the right part, the signal is sampled which leads to a discrete-time sample path with the sampling period T.

The previously introduced notions have to be differentiated from the system's dynamic that present both time-driven and event-driven behaviors. These notions are defined in Part 1 Section 1.1.2, and represent the mechanism based on which a system's state change (i.e.

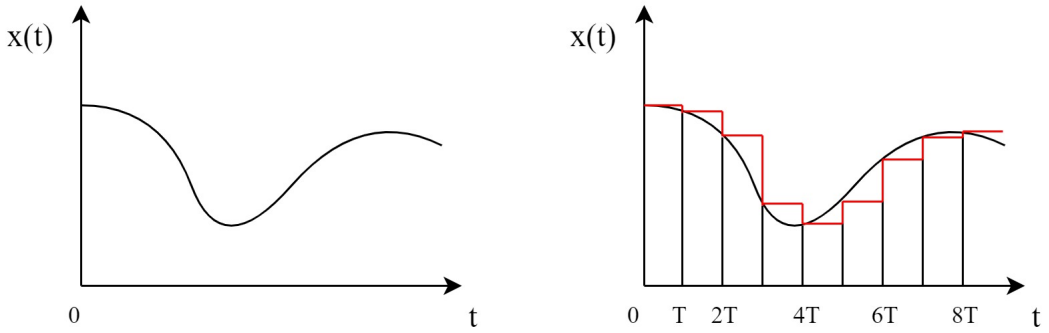

Figure 22: Continuous and discrete time representations of the same signal

state transition mechanism). Continuous-state systems are by nature time-driven and can be represented as continuous trajectories. Whereas discrete-state systems may be either time-driven or event-driven. Figure 23 synthesizes the different behaviors in a physical process.

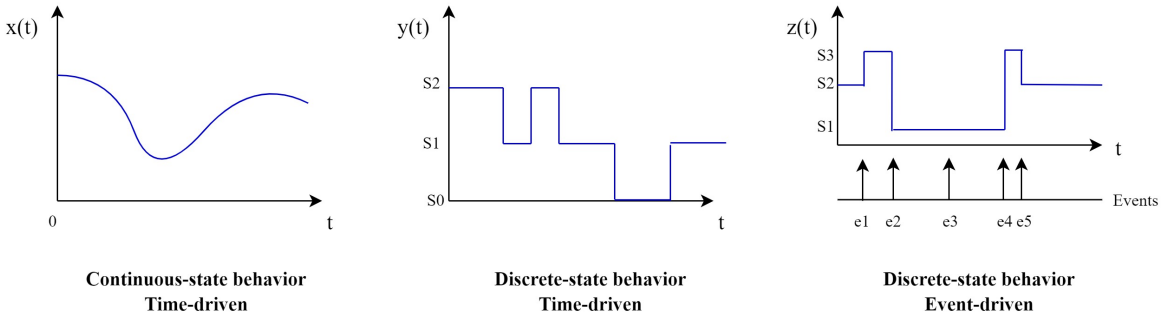

Figure 23: Example of continuous-state and discrete-state behaviors in a physical process

In our case, as we rely on a network-based detection, the observation trace is constituted of network packets containing information on the system's behavior. Therefore, as the network packets arrive successfully with their relative timestamps, our observation is discrete-time. As previously said, this discretization of time does not imply a discretization of the state space. Indeed, if we refer to the Figure 22 again, the signal $x(t)$ can still take any value in the ensemble $\mathbb{R}$ as in the continuous time case. Therefore, in our approach, we are able to observe process data presenting both continuous and discrete behaviors. Moreover, we are able to tackle time-driven and event-driven dynamics. The aim of our approach is to determine whether the behavior of the system is correct, i.e. define what are the correct trajectories for the process data (e.g. exceed a threshold, or not respecting a sequence of events). This task consist in characterizing the system's state.

B) General Approach

The overall concept of our intrusion detection approach is depicted in Figure 24. It has three steps: (1) Security Properties Extraction from international and industry standards in order to obtain a set of security properties. This step is detailed in Section 4.3; (ii) Security patterns synthesis which consists in translating security properties into security patterns. This step is explained in Section 4.4; (iii) Runtime Monitoring with the instantiation of runtime monitors and the actual monitoring against system's execution. This step is presented in Section 4.5.

First, we are going to provide definitions of some of the terms used in our approach. An *international standard* is a specification document. As stated by the International Electrotechnical Commission (IEC), international standards define *minimum requirements in terms of*

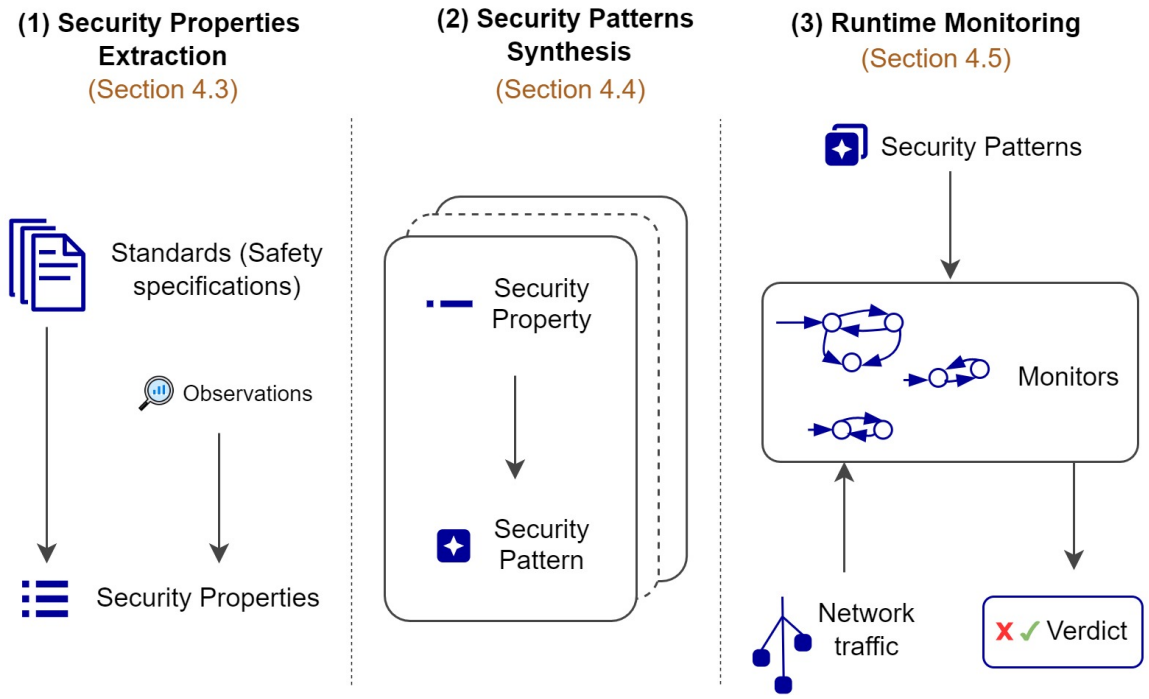

Figure 24: Overview of our approach

safety, reliability, efficiency and trust. Industry standards are a set of criteria within an industry relating to the standard functioning and carrying out of operations in their respective fields. Thus, international and industry standards provide the core safety specifications for a system.

IEC 61508 standard defines the *safety* of a system as the property to be "free of unacceptable risk" [85]. *Security*, on the other hand, relates to risks from malevolent actions (e.g., cyberattacks) that can impact the system itself in addition to its environment [100]. Therefore, security considers potential threats due to attacks while safety considers hazards and faults. *Security policies* are the rules that specify or regulate how an organization protects sensitive and critical system resources. Policies unambiguously state what is mandatory [89]. Our work considers only security policies violations specific to OT and which lead to safety violations. General IT intrusions that manifest otherwise (like Distributed Denial of Service (DDoS) attacks, for instance) are covered by complementary approaches.

Figure 25 summarizes the links between the previously introduced terms and shows the general idea driving our work. The starting point is a safety specification of the physical process which cannot be straightforwardly exploited for detection purposes. From this safety specification, we derive a security property that relates to the cybersecurity domain and has a meaning at a network level. The security property translates into a pattern that is exploitable for detection. The final step is the implementation of a monitor from this pattern.

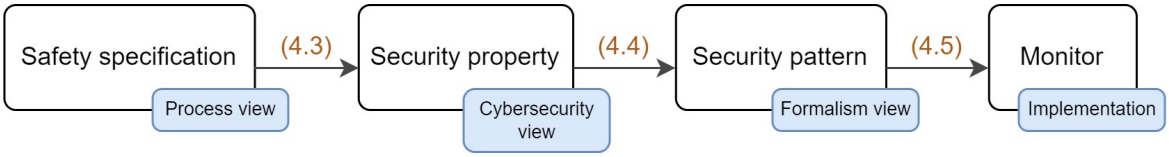

Figure 25: Main steps of a detection monitor synthesis

To illustrate the process in Figure 25, let us consider a robot that has two operating modes: (i) Moving and (ii) Homing. The Homing mode initiates a reference movement i.e. it generates a reference between a mechanical position and the actual position of the motor.

The steps from one safety specification to the corresponding monitor are the following:

a. **Safety specification.** From the standard, we get the following statement: "The operating mode Moving can only occur when the operating mode Homing ended successfully (i.e. once the system is referenced)."

b. **Security property.** In the network traffic it means: A movement order is observed in the network traffic only after observing the confirmation of the end of Homing.

c. **Security pattern.** Which translates as the pattern:
Precedence $(mode_{homing} \downarrow, mode_{movement} \uparrow)$.

d. **Monitor.** From the pattern, a monitor is implemented in Python code.

The output of every step, is detailed in next Sections; note that the corresponding Section numbers are represented in orange in Figure 24 and 25.

4.3 Security Properties Extraction

Concerning the sources for the extraction, three types exist:

a. **International and industry standards.** Most of the security properties are extracted from standards directly. The reason is that they contain the common characteristics of safe behaviors, since controllers and networks in ICSs conform with international standards.

b. **The device configuration related to the implementation of these standards.** Some security properties, though not directly extracted from standards, express the implementation of the aforementioned standards. They describe global specifications of the system.

c. **Network traffic observations.** The set of security properties is enriched with observations of the network traffic. This source of extraction provides use case dependant specifications.

Table 6 summarizes the security properties extraction sources, levels and types. For each of them, we detail the different types of properties.

Extraction source	Specification level	Type
International and industry standards	Local controller	Functioning states and modes
	Local loop	PLCs programming function blocks
	Network	General industrial protocol communication
Device configurations	Global system	Restrictions of standard functioning states and modes
Network traffic observations	Local controller, local loop, network or global system	Use case dependent properties

Table 6: Security properties extraction

INTERNATIONAL AND INDUSTRY STANDARDS

Local Controller

Device conformity with standards guarantees a certain level of safety and standard behavior. Manufacturer documentation provides details for specific implementations of standards. For instance, if we consider a servo drive, the IEC 61800 standards [86] specify its functioning states and operating modes as well as the corresponding mapping to communication protocols. The manufacturer documentation specifies the instantiation of standard parameters and

variables to memory addresses. Thus, in Figure 26, the Finite State Machine (FSM) of a IEC 61800 standard servo drive mapped to CANopen protocol, is represented. This FSM specifies the allowed transitions and events for each state.

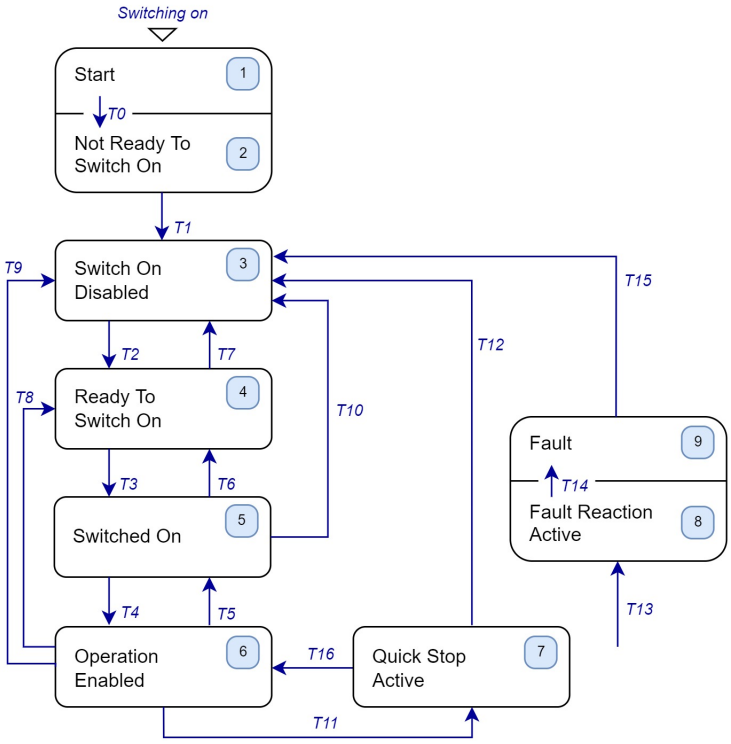

Figure 26: FSM for local controller specifications – Servo drive FSM from IEC 61800 standard

Examples are taken here for a servo drive, since it is a commonly used device with complex functioning. Similar operating specifications can be extracted for other devices. For instance, a pressure transmitter operating with CANopen would have some communication functions and operating flow, conforming with CANopen device profile DS-404 [33]. In the manufacturer manuals, further specifications are provided: supported CANopen objects, operating diagrams with different functioning modes, etc.

Local loop

This category concerns the control interface between PLCs and local loops and more specifically PLCs programming. PLCs are programmed in one of the languages defined by the IEC 61131-3 standard [84], i.e. Structured Text, Sequential Function Charts, Ladder Logic Diagram, Function Block Diagram and Instruction List. Reverse-engineering specifications from PLC programs is a difficult task with unreliable results, as implementation differs according to programmers' habits. Hopefully, for some usual tasks in control systems (such as communication, safety and motion control), standard library functions, based on standards like PLCopen [129] are used. Therefore, security properties effective between the distributed control layer and the local loop can be extracted.

As a matter of fact, many manufacturers are in compliance with PLCopen. For instance, standard function blocks libraries like Motion Function Blocks (MFBs) are provided for most automation platforms.

Network

This category is about network protocols and more specifically the application layer. Most of the time, the following types of networks can be found in ICSs:

- Fieldbus networks for the communication between PLCs and the local loops. A field-

bus is a real-time network connecting components through a common communication medium, also called "common bus" topology.

- Ethernet TCP/IP networks for the communication between local loops and the supervisory control.

DEVICE CONFIGURATIONS

Global system

This category concerns particular implementation of aforementioned standards. They are related to global system safety.

For instance, controllers can be limited to some functioning modes only. This means that in a given system, only specific operations are needed, and some controllers' operating modes may never be used. Thus, the extracted set of security properties is implementation dependant and limits the set of possible scenarios.

NETWORK TRAFFIC OBSERVATIONS

Local, global or network specifications

Some security properties may be discovered from network traffic observations. These properties can either concern local safety, global safety or network specifications. They are related to the specific use case and its architecture.

With this traffic monitoring, we infer all the specifications that are not entirely specified within standards. For instance, if a Homing command has to be issued from a standard PLCopen function block on two different motion axis, it will appear as two successive commands in the network traffic. The choice in the ordering (i.e. Homing the x axis first and then the y axis) is not specified by the standard as it relies on a programmer's choice. The exact property (i.e. order of Homing commands) is extracted from the network traffic.

4.4 Security Patterns Synthesis

In order to survey the system behavior, we need a formalism for specifications. Therefore, we translate the security properties into security patterns. The patterns are generalized descriptions [47] of requirements on the permissible state/event sequences in a finite-state system.

In Part I Section 2.2.4, we presented different classes of patterns that exist in the literature. In our work, we will rely on Dwyer patterns for patterns over qualitative time, Konrad patterns for patterns over quantitative time, and Maler patterns for quantitative time patterns extended to continuous signals. The great majority of the mixed behaviors of industrial systems can be expressed following such specification pattern systems. This is a great asset since it provides a finite number of well-defined patterns. Therefore, it facilitates the security pattern synthesis process and allows to reuse code for a specific pattern. Additionally, the fact that patterns are defined over scope eases the system specification process by simplifying their expression and permitting code reusability as well.

Expressing security properties as specification patterns is manually done in our work. Although not automatic, the security pattern synthesis process is systematic since we rely on predefined patterns and standards operation modes and requirements (see previous Section for details). Full automation (from the standards to security patterns) is not yet possible as standard specifications are generally not provided in a suitable format (like .xml) although some notable exceptions exist like the IEC 61850 standard [87]. Recently, it has been shown that partial protocol FSMs can be automatically extracted from Requests from Comments (RFCs) written in textual format [124]. It would be interesting to investigate this direction of work in order to generate automatically specifications as FSMs or security patterns, from international and industry standards.

4.5 Runtime Monitoring

4.5.1 Overall Process

To keep the IDS performance as close as possible to real-time, our approach is inspired from runtime verification, i.e. observing and evaluating the behavior of a system at runtime [17] (see Part I Section 2.2). Our monitors evaluate the security patterns previously extracted.

In order to be exploitable for detection purposes, the security patterns are mapped onto adequate formalism. In our work, we use the three previously introduced formalisms: LTL, MTL and STL to describe the variety of temporal behaviors of hybrid systems. LTL is adapted to express order or occurrence of events. LTL augments propositional logic with temporal operators. If the notion of range over time intervals is needed, MTL is more adapted. MTL extends LTL with time measurements over boolean signals. And for the case where real-valued (continuous) signals need to be described, STL is to be used. STL extends MTL over signals.

Dwyer patterns, may be mapped, for instance to LTL formulas. Konrad patterns, are expressed in MTL, while Maler patterns rely on STL (see Part I Section 2.2 for the presentation of these formalisms).

From the security patterns, monitors can be automatically generated. In runtime verification, the monitor is the object that constitutes the decision procedure for the property [17]. It verifies violations in system executions by checking the satisfaction of a given security property. The action of generating a monitor from a property is called the *monitor synthesis* and is further detailed below.

Monitors inherit scopes from the patterns, therefore at runtime the monitors shall be activated only within their scopes. In our approach, we introduce a module called the *Scope Recognizer* which activates and deactivates the monitors accordingly.

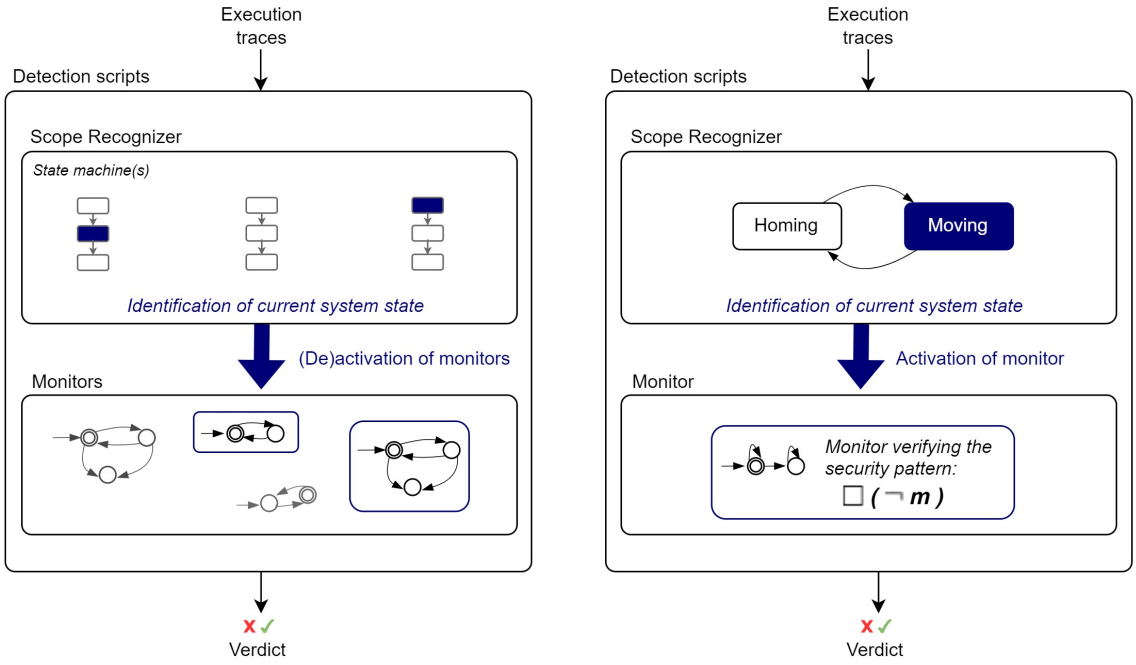

Figure 27: Runtime monitoring with the Scope Recognizer and monitors

The functioning of the Scope Recognizer is illustrated in Figure 27 (part a.). Scopes are related to the systems state. For instance, in a local loop composed of a servo drive, security patterns scopes are defined over the states of the FSMs in Figure 26. Operational modes of system components are taken into account as well. The Scope Recognizer identifies state

changes by monitoring status messages in the network traffic (i.e. in the execution traces).

The Scope Recognizer allows simplification of TL formulas. We take again the example of the robot having two functioning modes: Homing and Moving. One of the safety specifications says "no new movement (denoted m) order can occur while the robot is already moving". In other words, it means that "the event m should never occur while the system is in the functioning mode Moving". The corresponding security pattern is a Dwyer absence pattern *Never(m)* within the scope $Between(Mode_{moving} \uparrow, Mode_{moving} \downarrow)$. Expressed in LTL formalism, it gives the following complex formula: $\square$ $(((Mode_{moving} \uparrow) \wedge \neg (Mode_{moving} \downarrow \wedge \Diamond(Mode_{moving} \downarrow)) \rightarrow (\neg\ m\ \mathcal{U}\ (Mode_{moving} \downarrow))$.

In this example, the Scope Recognizer would monitor one FSM with two states: "Homing" and "Moving". Therefore, whenever the Scope Recognizer identifies the system as being in the state "Moving", the active scope detected is $Between(Mode_{moving} \uparrow, Mode_{moving} \downarrow)$. Then, the previously introduced LTL expression that has to be verified simplifies as: $\square(\neg m)$ (which corresponds to the Dwyer absence pattern *Never(m)* within a global scope) while the state "Moving" is active. This example is synthesized in Figure 27 (part b.).

4.5.2 Monitor Synthesis

In our work, we implement most of the monitors as FSMs, relying on an *automata-based* evaluation approach. When monitors take the form of a FSM, states are labeled by evaluation verdicts while transitions are labeled by predicates (events occurring in the systems).

First, "generic" FSMs are built offline for required patterns. The world "generic", here, implies that a FSM represents the tool able to evaluate a given pattern from the pattern systems previously described and that no process data is used yet. For instance, from Dwyer's *Occurence* patterns, let us consider the *Absence* pattern with global scope, which translates the fact that an event A never occurs. This pattern can be mapped onto LTL as the following formula: $\square(\neg A)$. The corresponding FSM is shown in Figure 28. In this FSM, the transition A has no physical meaning yet and could represent any event from the system.

Therefore, instances of generic FSMs are automatically created from the set of previously obtained security patterns. At runtime, a mapping of real process data is done on FSMs instances to evaluate corresponding TL formulas.

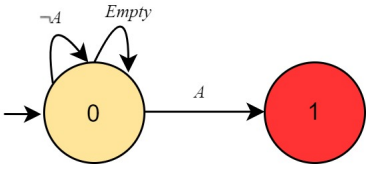

Figure 28: Generic Finite State Machine (FSM) for the *Absence* Dwyer pattern

To built generic FSMs, we use LTL_3 Tools[1], discussed in Part I Section 2.2.3. The tool identifies the generated states of the FSM with colors (red for "bad states", green for "good states" and yellow for "inconclusive states") depending on the truth value of the property to verify. Using this logic, "bad states" and "good states" are immediate and straightforward to interpret. However, "inconclusive states" are not. In Part I Section 2.2.3, we presented the FSM (Figure 10) generated from the LTL formula: $\square(Target_{reached} \rightarrow \bigcirc Motor_{stop})$. For easier reading, the FSM is shown here again in Figure 29. If state 2 is entered, an alert is immediately triggered, but if states 0 and 1 are entered, nothing happens since they are inconclusive states.

In our case, since we use the Scope Recognizer to detect context change, some monitors are activated and deactivated accordingly. Consequently, if a monitor is deactivated while its current state is an "inconclusive state", the verdict has to be interpreted. For this purpose, we

[1] https://ltl3tools.sourceforge.net/

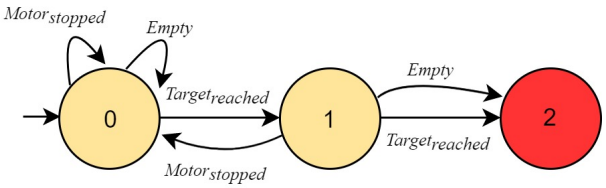

Figure 29: FSM generated by LTL$_3$ Tools for the formula $\square(Target_{reached} \rightarrow \bigcirc Motor_{stopped})$

use the notion of *presumably true* and *presumably false* truth values of the 4-valued logic [21] presented in Section 2.2.3. When a monitor is deactivated, the trace ends for this monitor. Therefore, whenever a monitor is deactivated, we evaluate the formula depending on the current state at the end of the trace. For instance, in our example displayed in Figure 30, if at deactivation, the current state is state 0, then the truth value is presumably true and no alert is triggered. Whereas, if the current state is state 1, the truth value is presumably false and an alert is triggered. This decision is based on the hypothesis that in the context of industrial systems, each formula is bounded by a specific context. For instance, in a given operating mode of a component, a request leads to an answer within the same operating mode.

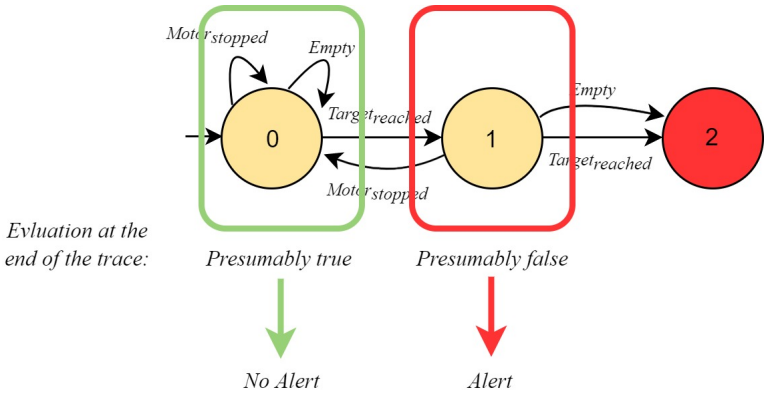

Figure 30: Verdict depending on the current state at the end of the trace, for the formula $\square(Target_{reached} \rightarrow \bigcirc Motor_{stop})$

Conclusion

Detecting process aware attacks in ICS requires to develop an intrusion detection methodology which takes into account the physical process. In our work, we consider process aware attacks aiming at damaging the physical process. In this Chapter, we presented a detection approach to efficiently monitor safety specifications of the system. The approach enables security properties extraction from international and industry standards. These security properties are translated into security patterns and monitored on execution traces of the physical process. The next Chapter details the implementation of the approach on an ICS. The approach is deployed and evaluated on a local loop.

Chapter 5

Detection for a Simple Hierarchical System

Contents

Introduction

In the previous Chapter, we presented an intrusion detection approach for process aware attacks relying on monitoring temporal specifications of system's physical processes. This Chapter aims at implementing the approach on a ICS testbed and evaluating it.

First, we present the use case setup for the experimental part of our work. The testbed we rely on is a cartesian robot which is a simple hierarchical system. Thus, we detail step by step the methodology to obtain the detection framework, as shown in Figure 25. Therefore, we provide concrete examples for the process from safety specifications to security patterns. We provide details about the corresponding monitors and implementation of our work. We also describe the software deployment of our architecture.

5.1 Use case Presentation - Cartesian Robot

In our work, we use a Schneider Electric industrial Cartesian robot, Figure 31, from G-ICS testbed (cf Section 3.4). Cartesian robots represent 40% of automatic handling solutions (Pick & Place). Such systems can be used to position raw material for a laser cutting machine for example. In our case, this robot is used to move a board below a pen. The overall system is a hierarchical and distributed ICS, composed of:

- A two axis positioning system MAXR12 Bergerlahr[1]
- Two actuators (motors) and two sensors (encoders), BSH[2] servo motors, used for each axis to control and detect limit positions.
- Two industrial servo-variable speed controllers, Lexium 32M[3].
- A PLC Modicon M340[4].
- A HMI Magelis XBTGT 2330[5].

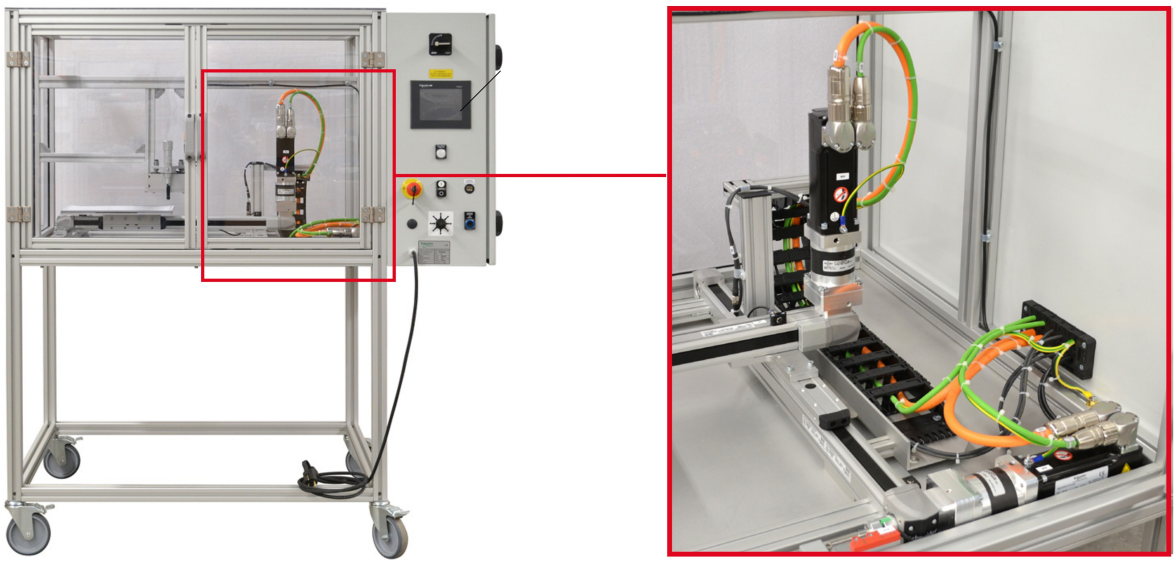

Figure 31: Two axis positioning system overview © All rights reserved

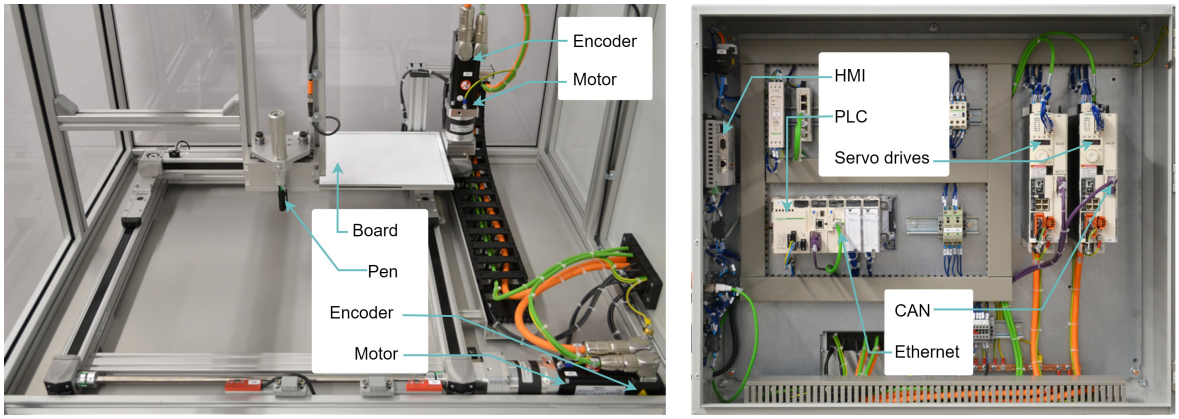

Figure 32: Two axis positioning system: Process view (left) and control system view (right) © All rights reserved

Two local loops are controlling the two motion axis. Each loop is controlled by the servo-variable speed controller. On each loop, the actuator and the sensor are directly wired to the servo drive analog inputs/outputs. The robot trajectory is controlled by the PLC either following a pre-defined trajectory or is manually driven by a human operator via the HMI.

[1] https://www.se.com/ww/en/product/MAXR12/portal-robot-lexium-max-2-directions-toothed-belt-roller-guide/
[2] https://www.se.com/ww/en/download/document/0198441113837-EN/
[3] https://www.se.com/ww/en/download/document/0198441113767-EN/
[4] https://www.se.com/ww/en/product-range/1468-modicon-m340/
[5] https://www.se.com/ww/en/download/document/35010372K01000/

The control components are identified in Figure 32. Communication between the PLC and the servo drives uses a CANopen fieldbus, while the HMI and the PLC are connected through a Modbus TCP/IP network. The overall hierarchical control of the use case is presented in Figure 33.

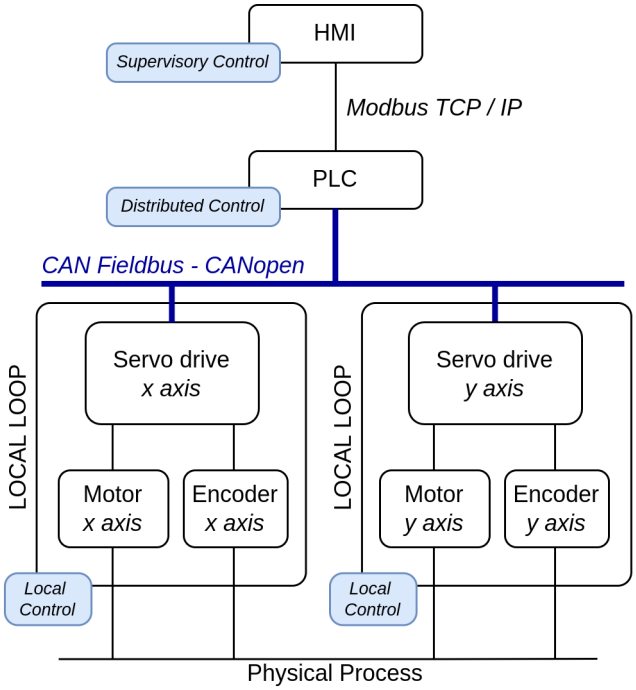

Figure 33: Use case hierarchical control

We consider this use case representative of hierarchical, distributed control, hybrid systems since the positioning control local loop is one of the most complex examples for local loops. Positioning control includes PID regulation with fast dynamics and event-driven control with several operating modes. Other types of local loops like temperature or pressure control will expose similar or simpler behaviors (less operating modes or slower dynamics). The trajectory control implies synchronization conditions between the different local loops. All the embedded control programs (HMI, PLC, local controllers) are provided by the manufacturer and certified to be conform with industrial standards.

5.2 Security Patterns Synthesis Process

In this Section, we detail the security patterns synthesis process, i.e. going from safety specifications to security patterns. For every safety specification source, detailed in Section 4.3, we give examples of some identified security patterns for our use case.

INTERNATIONAL AND INDUSTRY STANDARDS

Local Controller

Some safety specifications are directly related to the servo drive FSM previously presented in page 84 Figure 26. For easier reading, the FSM is shown here again in Figure 34. The only operational state from it is State 6: “Operation Enabled”. As previously stated, the states are observable from network traffic monitoring and used for the construction of the Scope Recognizer. The transitions are observable from network traffic monitoring as well, except for internally triggered transitions T_0 and T_{14}. The safety specifications with regard to the FSM are simple: within each state, only the transitions present in Figure 34 shall occur.

That gives a set of Dwyer occurrence patterns $Never(T_x)$ within the scope $Between(state\uparrow, state\downarrow)$, where T_x is a forbidden transition in $state$ and $\uparrow$ (resp. $\downarrow$) denotes the (de)activation of a state.

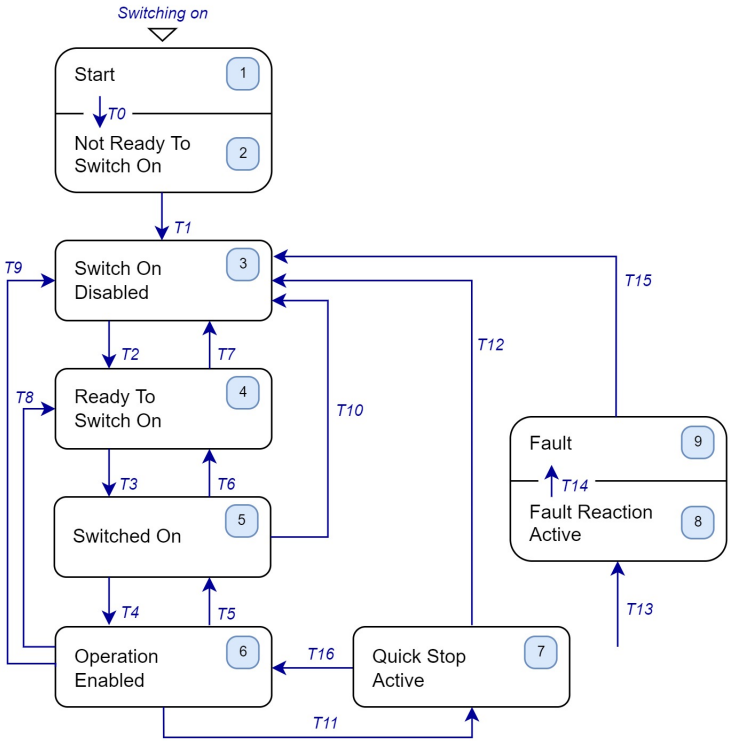

Figure 34: Servo drive FSM from IEC 61800 standard

Several operating modes are available when the servo drive is in "Operation Enabled" state (State 6 of Figure 34. As defined in [86], these modes include: (i) Jog (continuous movement), (ii) Homing (reference position), (iii) Profile Position (absolute or relative positioning), (iv) Profile Torque, (v) Profile Velocity, and so on.

Each operating mode has its own specific safety specifications. The exact list of operating modes supported by a specific servo drive is retrieved from manufacturer documentation. Our servo drive supports eight operating modes. In some modes, if the motor is running, changing mode is not allowed, e.g. in Homing mode, the corresponding security pattern is $Response(start_{homing} \rightarrow end_{homing})$ within the scope $Between\ (mode_{homing}\uparrow, mode_{homing}\downarrow)$. In Jog mode (continuous movement, i.e. driven by human operators), the end of a movement is signaled by two events: end of movement request and target reached. Then, the corresponding security pattern is $Response\ Chain(start_{Jog} \rightarrow end_{Jog} \rightarrow Target\uparrow)$ within the scope $Between(mode_{Jog}\uparrow, mode_{Jog}\downarrow)$. Still in Jog mode, a *Response Time* and a *Stabilization* Maler pattern for the movement speed can be identified. Similar patterns can be established for the other operating modes.

Local Loop

From PLCopen [129] standards, we extract security properties related to the function blocks used in the PLC programming. For instance, the MC_Jog function block timing diagram specifies the safe behavior of the continuous movement of an axis [129]. From this description, safety specifications can be deduced, e.g. "a new Jog command cannot occur while a Jog movement is active, or a Jog command specifying a movement simultaneously in forward (denoted fw) and backward (denoted bw) directions shall not be issued". This respectively translates into the security patterns $Bounded\ Existence(start_{Jog},1)$ with the scope $Until(Target\uparrow)$ and $Never(Jog_{fw} \wedge Jog_{bw})$ with scope $Between(mode_{Jog}\uparrow, mode_{Jog}\downarrow)$. Again, similar patterns

are deduced from the specifications of MC_MOVERELATIVE, MC_ MOVEVELOCITY and other function blocks described in the PLCopen standard.

Network

In our use case, the communication between the PLC and the local loops is made through CAN with a CANopen application protocol. The communicators in a CAN are called nodes. Distribution-wise broadcast is used with a publisher/subscriber model. That means that the communication between the PLC and the servo drives is not master/slave based like it is the case in most architectures (Modbus RTU for instance).

One of the specificities of CAN is that the frames do not contain the emitter address, but an identifier (CAN-ID) which indicates the priority of a frame. A node may use different CAN-IDs according to the criticality of the messages.

CANopen application layer adds an interpretation of the CAN frame fields (including the CAN-ID) suitable for control applications. CANopen defines messages (called objects) and priorities (CAN-IDs) for: network management (NMT), real-time data transfer (Process Data Objects – PDOs), configuration (Service Data Objects – SDOs), time synchronisation, time stamping, and device diagnostic. Numerous communication profiles were defined over CANopen objects in order to specify the behavior of common industrial control devices by the standard collection CAN in Automation (CiA). CiA 301 [32] gives the general NMT FSM specifying a CANopen node safe behavior, see Figure 35.

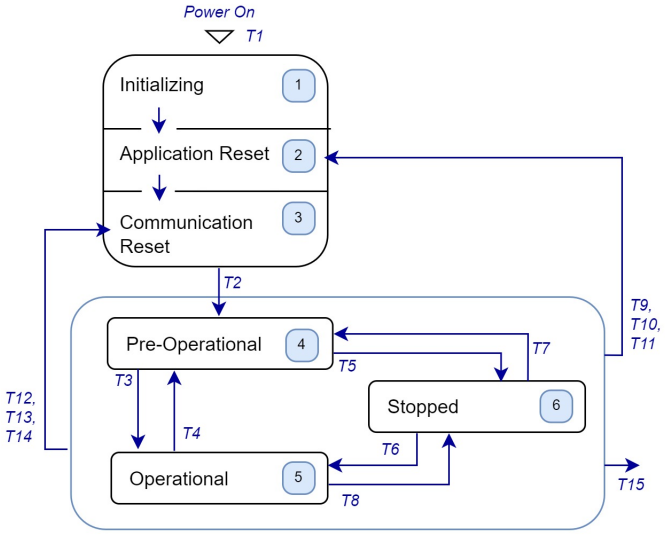

Figure 35: FSM for network protocol specifications: CANopen nodes FSM (NMT) from CiA standard

This NMT FSM (Figure 35) is independent of the servo drive FSM (Figure 34). This FSM adds new safety specifications, therefore new security patterns. The states Initializing, Application Reset and Communication Reset precede the CAN communication activation, therefore transitions between these states are not visible in the network traffic. All the other transitions in Figure 35 correspond to observable NMT messages. The security patterns corresponding to the safety behavior on the NMT FSM are also *Never()* patterns as the ones of the servo drive FSM (Figure 34).

A second category of network protocol-dependent security patterns are the ones concerning the real-time data transfer (PDOs). Although the basic CAN network operations do not attribute "master" and "slave" roles to specific nodes, it is necessary to define the "controller" (usually a PLC) and "controlled" (sensors and actuators) roles. This is achieved in CANopen using an adequate attribution of CAN-IDs. Seen from a controlled node, the real-time network traffic divides into Received PDOs (RPDOs) and Transmitted PDOs (TPDOs). RPDOs carry controls issued by the controller, whereas TPDOs carry sensors and states

information issued by a controlled node. TPDOs transmission is periodic. In our framework, the PLC plays the role of the controller and the servo drive the role of the controlled node. The servo drive will emit at fixed intervals a TPDO carrying the values of the current position and speed. A sensor false data injection frame will violate the periodicity of the samples. We use a variation of the Konrad's *Bounded Recurrence* Periodic pattern. The *Bounded Recurrence* denotes the "amount of time in which a state formula has to hold at least once" [96]. The new pattern we propose is called *Minimal Period* and is defined as the "amount of time in which a state formula has to hold at **most** once". Its MTL expression is $\Box(\Diamond_{(period,2*period)}TPDO \wedge \Box_{(0,period)}\neg TPDO)$. The periodicity value of this TPDO is retrieved from the CAN network configuration file.

DEVICE CONFIGURATIONS

Global system safety

In our use case, the restrictions of standard functioning states and modes are imposed by the HMI supervisory software. The corresponding specifications are:

- Only Homing, Jog and Profile Position (absolute and relative) modes are allowed. This constraint translates to *Never()* patterns for the occurrence of the other modes with Global scope.
- No movement is allowed if the system is not fully referenced. That means that right after the robot Powers On, a Homing cycle has to be performed on both axis prior to switching to any other operating mode. This translates into $Precedence(mode_{homing}\downarrow, mode_{Jog}\uparrow)$ and $Precedence(mode_{homing}\downarrow, mode_{Pos}\uparrow)$ with Global scope.
- Homing positioning occurs only once, then the pattern $Bounded\ Existence(start_{homing}, 1)$ with Global scope is added.
- Robot moves shall be contained within a range defined at the Supervisory Control level. This constraint translates into a Maler pattern of *Bounded* type for each axis position with scopes Jog and Positioning modes. Homing position is not concerned as the reference point is outside the robot working area.
- Axis operating modes are always synchronized. The easiest way to express this property in terms of security patterns is to use an *Universality* pattern, e.g. $Always(mode_{Jog_y}\uparrow)$ within scope $Between(mode_{Jog_x}\uparrow, mode_{Jog_x}\downarrow)$ to state that axis y has to be in Jog mode while x is in Jog mode.

The previous constraints are imposed by the embedded HMI software and are not modifiable by human operators. Other constraints like the motion speed, in one of the manual Positioning mode, may be dynamically modified by the system user. Therefore, such constraints cannot be handled by the current approach. However such dynamical variables can at least be constrained into intervals representing attainable limits.

NETWORK TRAFFIC OBSERVATIONS

Local safety, global safety or network

In our use case, the two servo drives have to obey to synchronization constraints. They are required to always be in the same functioning state. And in most of operating modes, they have to be synchronized as well. For example, if a Homing command is issued, it concerns both servo drives at the same time, which cannot be possible at the network level. Therefore, this operating mode synchronization constraint is translated into a sequence of two frames, each frame triggering the operating mode switching on, for each servo drive. The switching order is fixed by the PLC program specifications and can be discovered from network traffic. For instance, a *Precedence* switching order pattern with a Global scope is defined for each operating mode. Another example of specification extracted from traffic is about the current

screen displayed on the HMI. The current screen number is a PLC variable that appears in the traffic (read and write). Depending on this number, various options are selectable on the HMI by a human operator. Therefore, for each screen, only specific events can occur and only certain process variables are manipulated. Once again, the security patterns are *Never()* patterns within *Between()* scopes corresponding to the entering and exiting of HMI screens.

Another type of security patterns correspond to the hierarchical architecture of the system and they concern the comparison between inputs and outputs commands of the PLC. This means that every command that is addressed to the PLC on the Ethernet network will trigger several commands sent through the fieldbus. It also means that a command observed on the fieldbus has to be the consequence of a previous order issued at the Ethernet level. The corresponding pattern is a *Precedence* and *Response* pattern for every command. These types of security patterns survey the coherence of the PLC input data and its output data. For example, if a human operator select "Homing" on the HMI, the corresponding order will be passed on to the PLC which will issue the Homing command to the two servo drives. This example is represented in Figure 36.

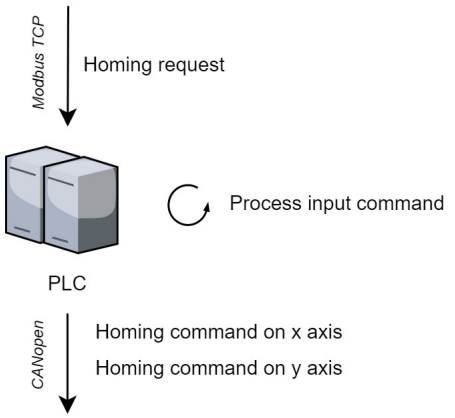

Figure 36: Coherence between inputs and outputs commands of a PLC

Table 7 synthesizes some of the security patterns used in our approach, with their formalism and corresponding formula to monitor. Note that each formula is given with global scope since the Scope Recognizer allows formula simplification.

Security Pattern	Formalism	Formula to monitor
[Dwyer] $Never(T_x)$	LTL	$\Box(\neg\ T_x)$
[Dwyer] $Response(start_{Homing} \rightarrow end_{Homing})$	LTL	$\Box(start_{Homing} \rightarrow \Diamond end_{Homing})$
[Dwyer] $Response\ Chain(start_{Jog} \rightarrow end_{Jog} \rightarrow Target\uparrow)$	LTL	$\Box(start_{Jog} \rightarrow \Diamond(end_{Jog} \wedge \circ(\Diamond Target\uparrow)))$
[Maler] $Response\ Time\ for\ movement\ speed\ s(t)$	STL	$\Box(start_{movement} \rightarrow \Diamond_{[0,0+t]}(\lvert s(t) - s_{ref}(t)\rvert < \epsilon_s))$
[Maler] $Stabilization\ for\ movement\ speed\ s(t)$	STL	$\Diamond_{[0,0+t]}(\Box(\lvert s(t) - s_{ref}(t)\rvert < \epsilon_s))$
[Dwyer] $Bounded\ Existence(start_{Jog}, 1)$	LTL	$(\neg start_{Jog}\ \mathcal{W}\ (start_{Jog}\ \mathcal{W}\ \Box\neg start_{Jog}))$[1]
[Dwyer] $Never(Jog_{fw} \wedge Jog_{bw})$	LTL	$\Box(\neg\ (Jog_{fw} \wedge Jog_{bw}))$
[Konrad] $Minimal\ Period$	MTL	$\Box(\Diamond_{(period,2*period)} TPDO \wedge \Box_{(0,period)} \neg TPDO)$
[Dwyer] $Precedence(mode_{homing}\downarrow, mode_{Jog}\uparrow)$	LTL	$(\neg mode_{Jog}\uparrow\ \mathcal{W}\ mode_{homing}\downarrow)$[1]
[Maler] $Bounded\ range\ for\ movement\ position\ x(t)\ on\ x\ axis$	STL	$\Box(x(t) < x_{max} \wedge x(t) > x_{min})$
[Dwyer] $Globally(mode_{Jog_y}\uparrow)$	LTL	$\Box(mode_{Jog_y}\uparrow)$

[1] with $\mathcal{W}$, the weak until operator. $p\ \mathcal{W}\ q = (\Box p) | (p\ \mathcal{U}\ q) = \Diamond(\neg p) \rightarrow (p\ \mathcal{U}\ q)$.

Table 7: Security patterns and formalism

5.3 Runtime Monitoring

In this Section, we detail the implementation of the detection approach. First, we present the way the data is captured, then we provide details on the Scope Recognizer and monitors

for our use case, and then our software deployment.

5.3.1 Data Capture

In this use case, the detection task relies on traffic capture between the PLC and the HMI as well as on the CAN fieldbus, as depicted in Figure 37. These network traffic capture points (represented in yellow in the Figure) do not deteriorate network traffic and they allow a monitoring of the system at runtime. These capture points allow to passively connect to the network and observe the traffic.

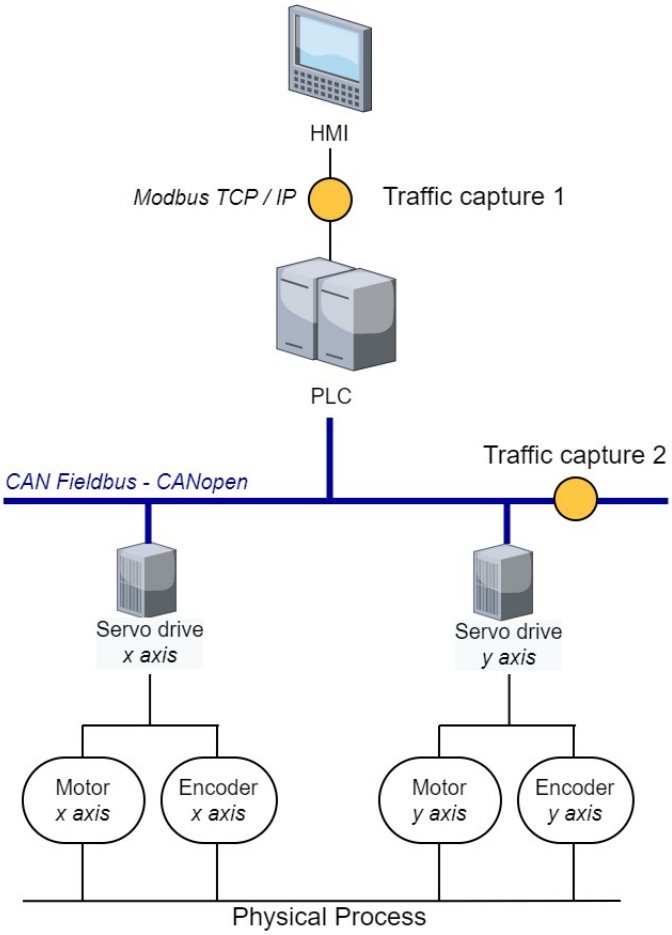

Figure 37: Traffic capture points of the use case

Depending on where the capture is performed, different technologies and equipment are deployed. They do not appear in Figure 37 since only their positions are indicated.

To capture network traffic on Ethernet-based networks (Traffic capture 1 on Figure 37), many technological solutions exist such as using port mirroring connection to a switch or Commercial-Off-The-Shelf network taps. Capturing network traffic on a fieldbus (Traffic capture 2 on Figure 37) is a real asset for anomaly detection since it gives access to data closest to the physical process. However, it is technically not as straightforward to implement as Ethernet level traffic capture. The architecture of the communication is simplified compared to traditional Ethernet-based networks. Indeed, considering the Open Systems Interconnection (OSI) model, only Layers 1 and 2 (sometimes Layer 7) are considered in fieldbuses architectures. Consequently, common tools used for Ethernet-based capture do not operate on fieldbuses. To address this issue, we used a PCAN-USB[6] adapter connected to the CAN fieldbus.

[6] https://www.peak-system.com/

At each traffic capture point, there is process data flowing and connecting local components. From this data, FSMs are constructed which translates the local context. The set of these FSMs constitute the Scope Recognizer.

5.3.2 Scope Recognizer and Monitors

At the CAN fieldbus capture level (Traffic capture 2 on Figure 37), for each servo drive, the **Scope Recognizer** encompasses 4 FSMs to activate adequate monitors:

- Servo drive FSM from IEC 61800 standard, represented in Figure 34.
- CANopen nodes FSM (NMT) from CiA 301 standard, represented in Figure 35.
- Servo drive operating modes machine inherited from IEC 61800 standard and Supervisory control specifications. In our use case, as stated in Section 5.2 part "Global safety", only Homing, Jog and Profile Position modes are allowed.
- A FSM with two states: "Movement" or "Motionless" for the physical process.

For simplicity's sake, these two last FSMs can be implemented as a unique FSM, represented in Figure 38.

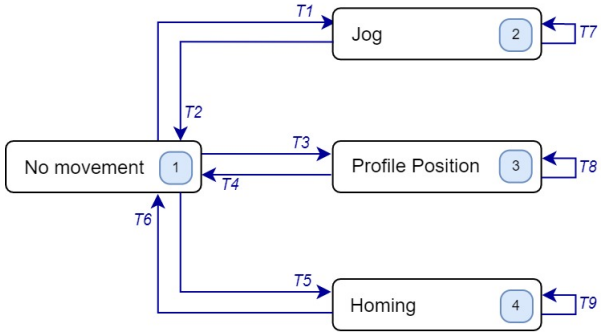

Figure 38: FSM of servo drive allowed operating modes

In total, 6 FSMs (3 for each local loop) identify the current behavior of the two servo drives from the aspect of internal state, network, operating modes and movement. The FSMs regroup every status information available in the network traffic.

Two additional FSMs are added to the Scope Recognizer, for Modbus TCP capture, between the PLC and the HMI:

- HMI screen FSM, which gives the current screen number displayed by the HMI. Such displayed screen numbers provide information concerning the supervisory context. There are 17 screens programmed in the HMI.
- Receipt FSM, which identifies the current set of operations, chosen by the human operator: a predefined trajectory or manual mode.

In total, the Scope Recognizer relies on 8 FSMs that are updated from traffic capture, at runtime.

In the Scope Recognizer, we store only the individual FSMs and we never compute the product of the FSMs since joint states are not used by the detection rules. Therefore, we keep CPU process memory usage low. Additionally, the Scope Recognizer permits to evaluate a state change only once, and store results that will be used for the activation of multiple monitors.

Concerning the **monitors**, the experimentation is performed with 33 monitors in total: monitors for verifying the correct transitions between states and modes of components, monitors verifying the operating of a state or mode, monitors verifying current position and speed, and monitors verifying the synchronisation of local loops. The number of monitors depends on the use case since the number of safety specifications is use case dependant. For example, for another local controller, the number of internal states and functioning modes is different, therefore the number of extracted security properties is different as well as the number of security patterns and synthesized monitors. We implement Dwyer, Konrad and Maler temporal patterns. The generalized security patterns allow reusability of the monitors code. For example, we have a lot of Dwyer *Never()* patterns to verify: there is only one monitor corresponding to this pattern in our detection scripts. We use it multiple times with different events or signals from the use case.

As previously introduced, in runtime monitoring, the evaluation of the TL formulas can be executed via two main techniques: *automata-based* [52] methods or *rewriting-based* [74] methods. We rely on automata-based techniques for the implementation of Dwyer and Konrad patterns. Concerning the evaluation of Maler patterns, we use a variant of the rewriting algorithm (the "incremental marking" procedure from [108]). With an automata-based technique, a monitor translates as a finite-state automata. For instance, the transitions of the automata are labeled by TL formulas; they are triggered by events or signals in the execution traces. In a rewriting-based monitor, the decision procedure is based on a set of rewriting rules triggered by a new event or signal. That is, given a TL formula and incoming events or signals, the monitor formula is rewritten into a new formula until a verdict (true or false) can be obtained.

Algorithm 1 shows the monitoring process for one monitor. Each time a network packet is received i.e. for each update of the observed execution trace, Algorithm 1 is executed for all monitors. This algorithm concerns the case of a monitor with an automata-based monitoring method. This means that the monitored pattern is used to generate and initialize a deterministic FSM. To keep the example more general, we do not detail a specific pattern here. Some tools exist to automatically generate the code for the FSM from a TL formula [22, 11]. In Algorithm 1, the variable *sm* is the state machine synthesized from the TL formula. Once initialized, this state machine has generic transitions named $transition_1 \cdots transition_n$. These transitions are triggered when an event of interest $e_1 \cdots e_n$ appears in the network traffic. Note that the algorithm encompass both sequential and event-driven programming. In the sequential part, *sm* is updated depending on the flow of incoming execution traces. Whereas the event-driven programming depends upon the entering and the exiting of specific states of *sm*. Therefore, based on the evolution of current states, some methods are called to check if the pattern is expired or if an invalid transition is triggered inside the state machine. As soon as the verdict v is set to false, an alert is raised. Note that in the presence of the timeout variable, the evaluation of time (function $sm.check\_time()$) is always done with the timestamp of frames. This allows our framework to tackle communication delays and latency in the arrival of events.

5.3.3 Software Deployment

We rely on the open source NIDS Zeek[7] v4.1.0-dev.704 for the deployment of our IDS framework. The rationale for using Zeek in our work is provided in Section 3.5 page 74. The main reason for this choice is that Zeek is able to raise not only alerts but also general events (like a packet arrival) allowing the implementation of security pattern monitors. Unlike most other open-source IDS, Zeek is highly configurable and admits the development of sophisticated rules in its own scripting language.

[7] https://www.zeek.org

Algorithm 1: Monitoring process for an automata-based monitor

Data: t: observed execution trace
p: security pattern (TL formula)
$e_1 \cdots e_n$: events of interest into the network traffic
active: monitor's attribute set by the Scope Recognizer
timeout[optional]: interval timeout for an event
Result: v: verdict

```
/* SEQUENTIAL - iterate over all monitors */
sm ← initialize(p) ;                                   // initialize state machine
v ← True ;                                  // initialize verdict to "satisfied"
while received(t) not null do                          // new execution trace received
    ts ← get_ts(t) ;                    // get observed execution trace's timestamp
    if active == 1 then
        if is_event(e_1, t) == 1 then                          // update state machine
            sm.trigger(transition_1)
        if is_event(e_2, t) == 1 then
            sm.trigger(transition_2)
        ...
        if is_event(e_n, t) == 1 then
            sm.trigger(transition_n)
```

```
/* EVENT-DRIVEN */
if sm.check_time(ts, timeout) then                     // if pattern is expired
    v ← False
if sm.invalid_trigger() then                 // if invalid transition is triggered
    v ← False
return v
```

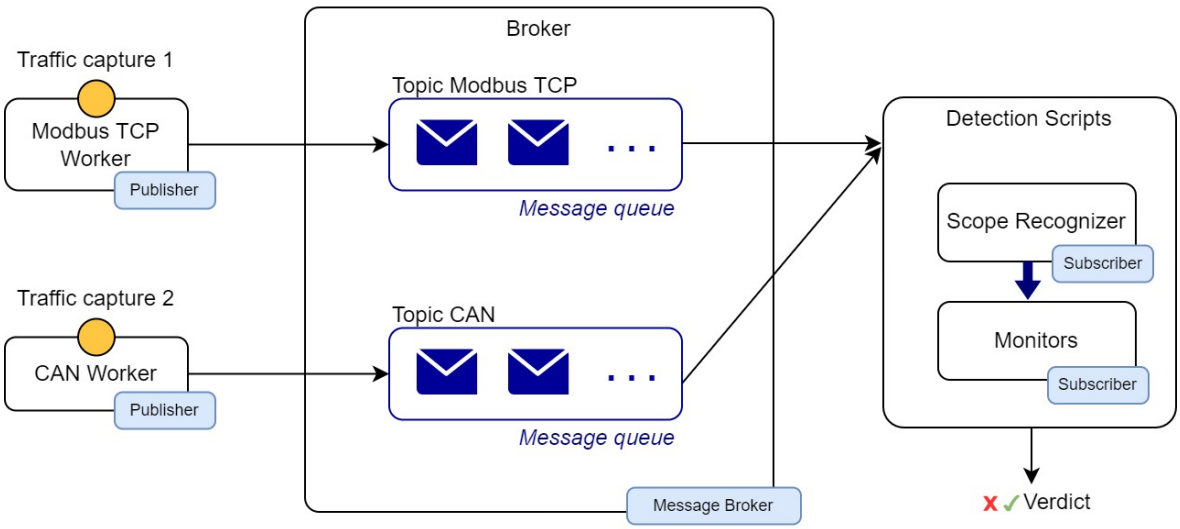

Figure 39: Architecture of our Standard Specifications-based IDS

The overall architecture of our detection framework is presented in Figure 39. The Workers are software agents which intercept the network packets and filter network traffic. They publish messages (alerts or general events) to a Broker agent which centralizes and dispatches the messages. In our detection scripts, the Scope Recognizer subscribes to different topics and manages the activation/deactivation of monitors depending on the current state of the system (i.e. depending on the current scope). Monitors subscribe to certain topics as well and give verdicts. The publisher/subscriber protocol uses the "Broker" Zeek framework which extends C++ Actor Framework[8].

As explained in Section 3.5 page 74, Zeek is limited to Ethernet-based protocols. The only industrial protocol parser available in Zeek is Modbus TCP. For our study, we extended the Zeek framework to fieldbuses. We developed a Zeek CAN module based on SocketCAN[9] kernel module for frame capture. We also developed the Broker based on the Zeek broker framework and implemented the Scope Recognizer together with its FSMs and the diverse monitors in the detection scripts.

The Broker and detection scripts were implemented in Python 3.8.10. while CAN Worker is developed in Zeek native language. The framework runs on a Dell Latitude 5414 with a 2.6 GHz Intel Core i7-6600U processor 6GB 2133 MHz memory under Ubuntu 20.04.3 LTS. Extract of code is made available[10].

5.4 Evaluation

We evaluate our approach on different aspects:

- **Detection capabilities.** (Section 5.4.1) This part verifies the correctness and effectiveness of the approach through time performance evaluations and other key factors for intrusion detection methodologies.
- **Scalability.** (Section 5.4.2) This part evaluates the scalability of the approach to large ICSs.
- **Extensibility.** (Section 5.4.3) This part discusses the generalisation of the approach and its usage on other use cases.

5.4.1 Detection Capabilities

As presented in Section 5.3, our detection framework relies on 33 monitors. We evaluate it against different categories of attacks. To have a full benchmark coverage, all our monitors are triggered throughout our experiments.

Attack implementation. We deployed attack scripts that send commands on the CAN fieldbus and Modbus TCP network that attempt to bring the process to an unsafe state. An attack is implemented as a Python script. On the CAN fieldbus, we use a PCAN-USB[11] adapter connected to the fieldbus. On the Modbus TCP network, we implemented attacks with a Modbus TCP client using PyModbus[12] Python tool.

Attack detection and response time. The consequences of attacks can be visualised on the physical process. Some attacks will disrupt the positioning process but without other consequences, while other attacks will put the servo drive into a permanent fault condition requiring a manual reset. We evaluated representative attacks that could occur against our use

[8] https://www.actor-framework.org/
[9] https://www.kernel.org/doc/html/latest/networking/can.html
[10] https://github.com/V3Hr7LnNRu7T/SSIDS
[11] https://www.peak-system.com/
[12] https://github.com/riptideio/pymodbus

case. We grouped attacks by categories, see Table 8. For example, let us take the first category of the table: "Unauthorized transition of NMT states". This category of attacks concerns either: (i) sending PDOs during Pre-Operational state (State 4 of Figure 35) or (ii) sending any other command than NMT related ones during Stopped state (State 6 of Figure 35). Therefore, in this category, we consider that there are 2 different attacks. In a similar manner, attacks from the category "Movement command outside authorized positioning zone" can be performed through the drive mode Profile Position using absolute or relative positioning but attacks can also be done through Jog movement. Table 8 summarizes: an identification number (ID) for the different categories of attacks, the categories of attacks, the number of related attacks in each category, the attack entry point, the physical effect on the system, the Worst Case Response Time (WCRT) and the Average Response Time (AvRT) for the detections. The results in Table 8 were obtained for 100 executions of each attack. In total, we considered 93 attacks of 22 generic categories. The majority of attacks were implemented on the fieldbus, as some attacks can only be implemented at a direct proximity to the physical process (e.g., for sensor false data injection).

Table 8: Implemented attacks and their effect on the system

ID	Category of attack	# Attacks	Entry point	Physical effect	WCRT (ms)	AvRT (ms)
	Forcing state transitions					
1	Unauthorized transition of NMT states	2	Fieldbus	Drive: permanent fault condition	15.5	9.7
2	Unauthorized transition of drive states	16	Fieldbus	Depends on states	14.9	9.6
3	Unauthorized drive mode	8	Fieldbus	Disrupts the positioning process	19.0	12.2
4	Hijack drive modes (change from one to another)	6	Fieldbus	Disrupts the positioning process	13.7	9.4
5	Unauthorized transition of drive states	4	Modbus TCP	Depends on states	22.0	9.9
	Illegal command within a state					
6	Unauthorized commands in specific NMT state	2	Fieldbus	Drive: stopped	16.7	10.4
7	Unauthorized commands in specific drive states	6	Fieldbus	Depends on state	14.4	8.2
8	Unauthorized commands in specific drive states	6	Modbus TCP	Depends on state	21.9	10.1
	Movement command injection					
9	Movement command while the system is moving	3	Fieldbus	Disrupts the positioning process	14.7	8.3
10	Movement command while the system is motionless	3	Fieldbus	Disrupts the positioning process	18.0	7.9
11	Movement command when the system is not referenced	3	Fieldbus	Disrupts the positioning process	17.5	8.0
12	Movement command outside authorized positioning zone	3	Fieldbus	Drive: permanent fault condition and disrupts the positioning process	12.3	6.9
13	Illegal Jog directions	2	Fieldbus	Disrupts the positioning process	11.9	7.5
14	Unauthorized speed for movement	3	Fieldbus	Disrupts the positioning process	14.9	6.8
15	Movement command while the system is moving	3	Modbus TCP	Disrupts the positioning process	18.4	9.4
16	Movement command when the system is not referenced	3	Modbus TCP	Disrupts the positioning process	14.5	9.4
17	Movement command in multiple directions simultaneously	11	Modbus TCP	Disrupts the positioning process	20.9	10.4
	Sensor false data injection					
18	Periodicity of TPDOs (system moving and motionless)	2	Fieldbus	Disrupts the positioning process	14.8	7.7
	Illegal sequence of operations					
19	Homing command on y axis first, then x axis	1	Fieldbus	Disrupts the positioning process	13.8	8.1
20	Homing command on one position axis only	2	Fieldbus	Disrupts the positioning process	12.9	7.5
21	Homing command when the system is already referenced	2	Fieldbus	Disrupts the positioning process	14.7	7.2
	Targeting synchronisation of local loops					
22	Each drive in a different operating mode	2	Fieldbus	Disrupts the positioning process	16.6	7.2

In summary, we detect attacks that aims to: interrupt current operations, move the robot outside the safe area or with an unsafe speed, force the change of the internal state or operating mode of the speed drive or disrupt fieldbus and Modbus TCP communications. We detect the side effects of attacks on the system. It means that the launching point of the attack does not matter, whether fieldbus or Modbus TCP network. Thus, an attack triggered by corrupted code inside a PLC could also be detected. Concerning the response time for the attack detection, the WCRT is never above 22.0 ms. The mean value for the response time is around 10 ms. For the attacks using Modbus TCP as entry point, they globally have a higher response time. This can be explained by the fact that in addition to the time required for the detection, we need to add one entire cycle of PLC which processes the command of the

attack. The cycle time of the PLC implied in the attack is 10 ms.

False negatives and false positives. Our IDS detects all the aforementioned attacks as they were forged to target the security properties previously defined. Therefore, we have no false negatives, confirming the correctness and effectiveness of our monitors synthesis.

Some false positive alerts were observed due to a configuration issue of the CAN nodes. During Pre-Operational, Operational and Stopped states of the CANopen FSM depicted in Figure 35, each node periodically broadcasts its state ("heartbeat" messages). The heartbeat period is configurable by the user. If it is configured too long, a node may switch between several states before broadcasting its state, as heartbeat messages are exclusively periodic and not triggered by the FSM transitions; e.g. at Power On, a node may go through Initialization, Pre-Operational, Operational and emit a TPDO before issuing the first heartbeat about being in Operational state. A monitor raises an alert as PDOs are forbidden before Operational state. Although this alert does not correspond to an attack, it is corresponding to a violation of the standard specification due to a misconfiguration of the device.

Alert overlapping. A single network packet may trigger several monitors. For instance, if the robot is moving in Jog mode, a Positioning command with a target outside the safe area and a speed setpoint superior to the maximal admitted will trigger four alerts: illegal operating mode change, Jog movement not ended, illegal destination and illegal movement speed. The four security patterns are clearly not redundant but it is impossible to link the four alerts to the same attack. The development of an alert correlation methodology would be needed in order to reconstruct the attack scenario in such cases.

5.4.2 Scalability

In order to assess the scalability of our approach, three performance measures are important: (i) Response time for various attacks, already evaluated in Section 5.4.1, (ii) the execution time of the detection loop and (iii) the maximal number of local loops that can be monitored with a single instance of our detection framework.

Execution time of the detection loop. Having an on-line monitoring approach, the most important measure is the computational time of a full monitoring step. For every received network packet, each active monitor has to perform an update step. The addition of all these update steps is therefore the total execution time of the detection loop. The result depends on the number of active monitors, and complexity and type of patterns (Dwyer, Konrad, Maler).

Active monitors	1	5	10	20	30	50	80	130	200	300	450	600	800
WCET (ms)	1.09	1.15	1.19	1.17	1.19	1.30	1.66	1.49	2.06	2.08	2.59	2.98	4.29
AvET (ms)	0.80	0.82	0.84	0.87	0.89	0.95	1.04	1.17	1.36	1.67	2.08	2.50	3.07

Table 9: Impact of the number of active monitors on the execution time of the detection loop

We gradually increased the number of active monitors by duplicating one of the most complex (in our case, this is the "Absolute Position" monitor). The detection loop computer program is running with high priority. The Worse Case Execution Time (WCET) of the detection loop and the average execution time (AvET) of the detection loop are represented for 1, 5, 10, 20, 30, 50, 80, 130, 200, 300, 450, 600 and 800 active monitors, see Table 9. Each AvET is the average of the timings of 2000 executions. The WCET and AvET of the detection loop are represented in Figure 40.

LTL formula evaluation is known to be a polynomial complexity problem [131]. The algorithm for the monitors is of polynomial complexity as well as it is proportional to the number of local loops. As expected, the curves for WCET and AvET of the detection loop are poly-

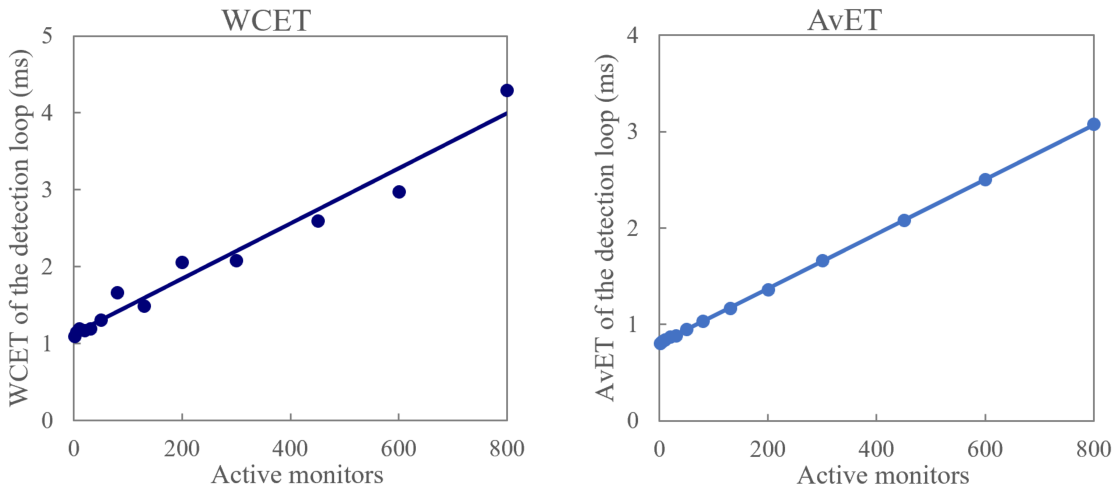

Figure 40: Worst Case Execution Time (WCET) and Average Execution Time (AvET) of the detection loop depending on the number of active monitors

nomial, more precisely linear.

Maximal number of local loops. As we are doing on-line detection, this limit depends on the execution time of the detection loop. As the number of monitored loops increases, the fieldbus traffic intensifies and the number of active monitors in the detection loop increases. On-line detection requires that the detection loop execution time stays inferior to the packets inter-arrival time. We compare the WCET previously obtained with the expected inter-packets arrival time.

The network throughput in our experiments is the following:

- On the CAN fieldbus, at most 120 packets/second[13]
- On Modbus TCP Network, at most 100 packets/second with mainly "reading" functions. Indeed, whenever a HMI screen is displayed, some specific variables are periodically read.

A monitor may subscribe to both networks (CAN fieldbus and Modbus TCP network), therefore the total input throughput for this monitor is the addition of the two individual throughput, i.e. 220 packets/second (inter-packets arrival time of 4.5 ms). This example constitutes the worst case for the input throughput of a monitor. In our use case, the detection loop WCET is around 1 ms for 30 monitors (see Figure 40), which is more than four times inferior to the inter-packets arrival time.

Now, let us compute the maximal number of local loops. We take our use case as a reference, therefore we consider that 33 monitors correspond to 2 local loops. With this hypothesis, we evaluate the WCET of the detection loop depending on the number of loops. Then, in order to calculate the inter-packet arrival time depending on the number of loops, we consider that every local loop generates 60 packets/second[14]. The results are represented in Figure 41.

We can conclude that one instance of the detection scripts can handle the monitoring of up to 11 position control local loops. Those experiments are achieved with a single Zeek Broker instance. If more detection capacities are required (more local loops and monitors), the

[13] The network throughput peak appears during movements with TPDOs every 20 ms (i.e. 50 TPDOs/second per drive; 100 in total), heartbeats every 500 ms (i.e. 2 heartbeats/second per drive; 4 in total) and other network protocol functions, such as "Sync", SDOs, etc.

[14] 120 packets/second for 2 local loops, as aforementioned.

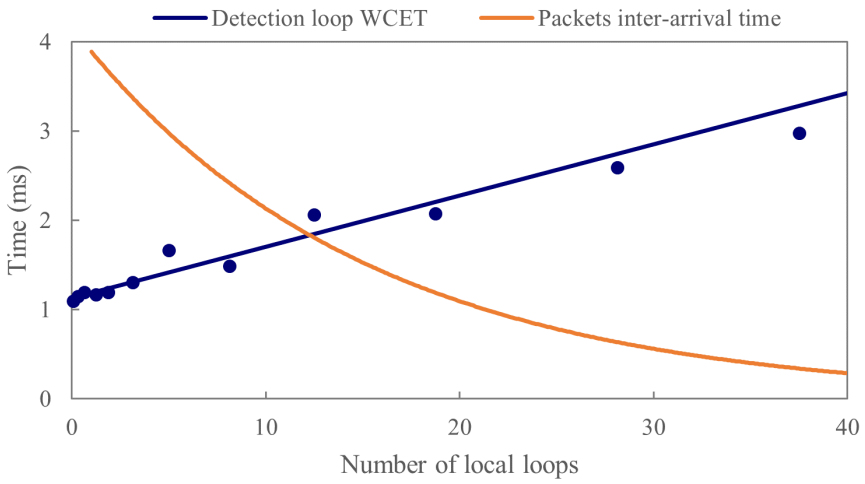

Figure 41: Maximal number of local loops

approach can be easily extended based on the Zeek framework and our Workers and Broker. Several Brokers could filter network traffic and redirect network packets to different detection scripts.

5.4.3 Extensibility

Since extensibility is about the generalisation of the approach and its usage on other use cases, we decided to implement our framework on a second ICS testbed.

A) Deployment of the framework on the naval testbed

Therefore, a similar work to the one described in the present Chapter has been deployed on the naval testbed presented in Chapter 3.4.

The deployment of our approach on this testbed concerns the "Direction" subsystem loop of the testbed (see Figure 20). Network traffic capture is achieved at fieldbus level as well as between a PLC and an HMI; as illustrated in Figure 42.

The differences between this naval use case and the cartesian robot use case are multiple: the equipment are different and the implemented fieldbus protocol as well. The cartesian robot's fieldbus relied on CANopen protocol whereas the naval use case implements Modbus RTU. As for the cartesian robot, we continued the extension of the approach to other fieldbus protocols. A stand-alone Modbus RTU Worker was developed using C and C++ languages. Note that it this use case, the two servo drives are not in the same fieldbus.

As for the previous use case, traffic capture at high levels network (Traffic capture 1 on Figure 42) is more straightforward than fieldbus traffic capture (Traffic capture 2 and 3 on Figure 42). To do the latter, we use RS-485 to USB converters[15] together with a stand-alone Modbus RTU worker[16] and Zeek. Figure 43 shows the set up we rely on for network traffic capture on one fieldbus. The first RS-485 to USB converter is used to capture every network packet that flows on the fieldbus while the second converter is used for sending attacks on the bus.

Concerning the implementation of the framework, the Scope Recognizer could be deployed almost identically as the one developed for the first use case. Indeed, the servo drives, although different in brand, follow the same state machine defined by the IEC 61800 standard

[15] https://joy-it.net/en/products/SBC-TTL-RS485
[16] Codes can be found in the GitHub https://github.com/V3Hr7LnNRu7T/SSIDS

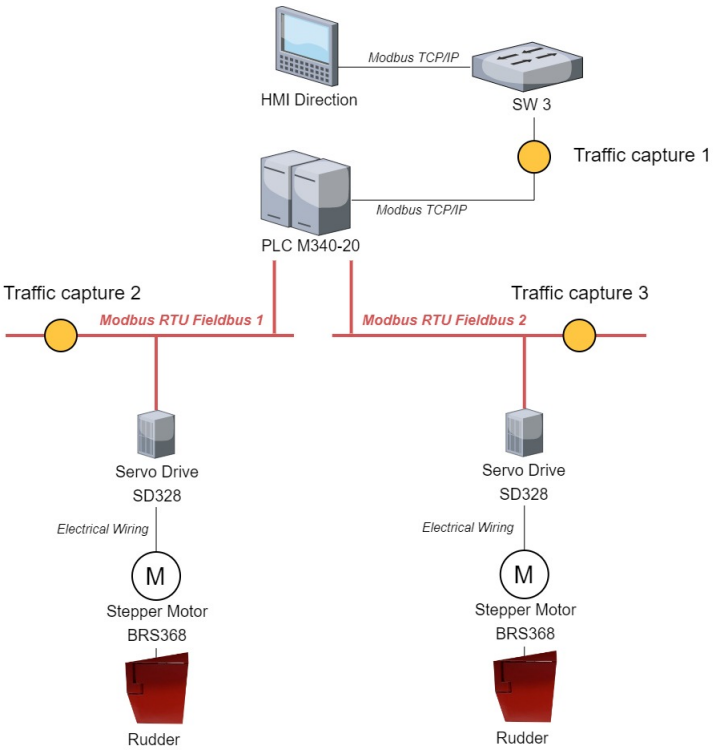

Figure 42: Naval testbed and traffic capture points

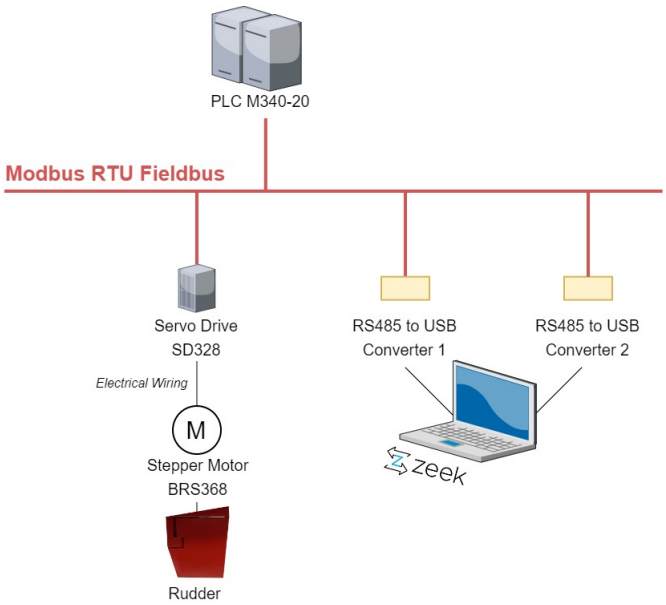

Figure 43: Set up for fieldbus traffic capture

(Figure 34). However, in the Scope Recognizer, an adaptation of this FSM has to be done for the mapping to Modbus RTU protocol. Additionally, there are no equivalent of the CANopen FSM for Modbus RTU, therefore the Scope Recognizer encompasses one less FSM. Furthermore, the use case dependant FSMs have to be adapted. For instance, the authorized servo drives mode are more restricted in this use case as only *Profile Position* functioning mode is used. Also, the FSM for the HMI pages is quite smaller, since the HMI presents only 3 pages.

In total, 21 monitors were deployed including 2 monitors concerning synchronisation of the two servo drives. In this use case, only 3 types of security patterns were used: *Precedence*, *Response* and *Absence*. The monitors where implemented using Reelay[17] library [151]. This library relies on Past-MTL with the temporal logic operators: *previously* (analogous to *next* in the past), *once* (*sometime* in the past), *historically* (*always* in the past) and *since* (*until* in the past).

Concerning the evaluation, we compute the execution time of the detection loop. Once again, the idea is to measure the computational time of a full monitoring step. Due to the few security patterns used in the implementation (3 security patterns), the evaluation was conducted on each of them (i.e. on a monitor evaluating the pattern). The results are shown in Figure 44. For each type of pattern, the curves show a similar trend as the one presented in Figure 40: the execution time of the detection loop depending on the number of active monitors is linear.

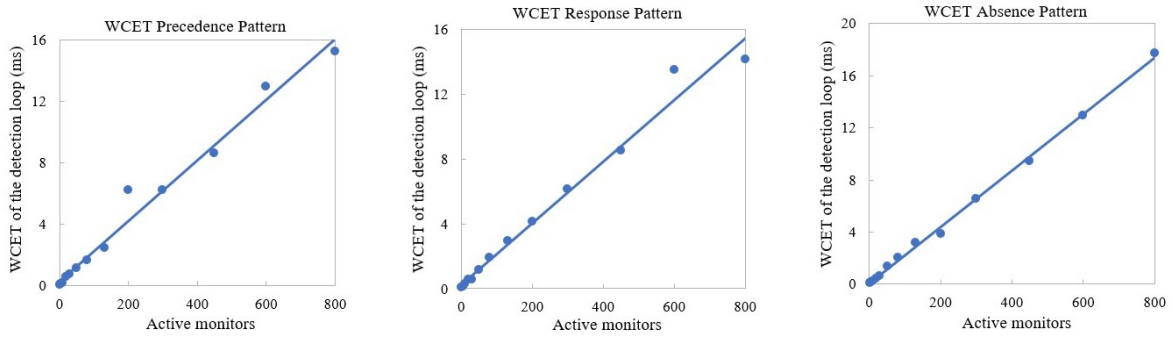

Figure 44: Execution time of the detection loop depending on the number of active monitors – Evaluation for 3 security patterns

However, if we compare the WCET obtained in the first use case (see Figure 40) and the one from the second use case evaluation, there is almost a 4.5 ratio (for the Absence Pattern). The fact that the first case implementation showed a 4.5 times faster execution time can be explained by different factors: (i) implementation of the decision procedure (in the first case, it was decomposed with some manual steps, whereas the second was used in a blackbox manner), (ii) TL formalism considered (LTL, MTL, STL in the first case, past-MTL in the second), (iii) system under observation having different network characteristics and (iv) characteristics of the computer conducting the evaluation. In conclusion, despite this difference in the execution time of the detection loop, the experimentation has shown that the framework can be quickly and easily extended to different use cases.

B) Summary on extensibility

Multi-protocol support. The approach can be extended to any fieldbus protocols. In the first use case (cartesian robot) the approach is deployed for CAN protocol whereas in the second use case (naval testbed) the Modbus RTU Worker allows the deployment of the framework on serial Modbus lines. The detection patterns are the same except for, of course,

[17] https://github.com/doganulus/reelay

the protocol-dependent patterns.

Multi-component support. On the use of different local controllers, the same steps can be implemented for the approach. We considered one of the most complex use case possible for a local controller (i.e. servo drives, for both use cases). Similar behaviors could be extracted from other local controllers.

Reusability. Additionally, we do not have to start over the specification analysis for a new system. Once extracted from an international or industry standard, the safety specifications are the same for all the controllers from the same category: servo drives, temperature or pressure regulators, PID controllers, etc. No matter the manufacturer, a controller conforms with standards. We limit the information extracted from traffic and control logic to parameters of the standard specifications. To that extend, we argue that the security properties extraction process is easily replicable from one system to another.

Conclusion

This Chapter presented the implementation of our approach framework for efficiently monitoring security properties. We presented a use case and deployed our framework onto it. Our approach is deployed on the communication network between the distributed control and the local loops and between the supervisory control and the distributed control. The security properties for the monitoring are systematically retrieved from international and industry standards, or infered from traffic. We note that the security properties need to be retrieved from the standards only once for each type of device as they are common among all the manufacturers compliant with the standard.

There are several original points in our contribution. We are able to handle hybrid process dynamics with a scalable, low complexity approach. We are able to perform detection at the lowest network level, i.e. fieldbus level, which is the closest to the physical process. We do not need large and representative datasets to construct our model and we do not rely on heavy assumptions such as periodical system behaviors. As a matter of fact, in most of existing approaches, legitimate operator manual interventions trigger alerts since they are breaking the supposed periodic behavior of the system. It is not the case of our approach, thus we are able to handle human-driven systems.

The approach was evaluated on two ICS physical testbeds. The evaluation of the approach has shown its correctness and effectiveness through time performance evaluations. In total, 93 types of attacks were conducted from both fieldbus and TCP networks and successfully detected. Furthermore, the approach was shown to be scalable and extensible.

The evaluation of the approach has shown promising results on a simple hierarchical system. The next Chapter investigates a distributed deployment for the approach.

Chapter 6

Distributed Detection

Contents

Introduction

In the previous Chapter, we presented our specification-based IDS based on monitoring security properties on the network traffic for simple hierarchical systems. We evaluated our approach for one instance of the detection scripts. However, if more detection capacities are required in terms of number of monitors (for a larger use case for instance), the approach can be distributed on several machines to balance intrusion detection workload. Nevertheless, this inevitably introduces only local views on the system. The challenge is to manage these local views in order to monitor global security properties of the system. This is the topic of the current Chapter.

First of all, we present the concept of our distributed IDS approach. Then, in a similar manner as the previous Chapter, we provide concrete examples for a distributed deployment of our intrusion detection methodology. Relying on a bigger use case, we provide details about the corresponding monitors and implementation of our work. We also describe the software deployment of our architecture and evaluate the approach.

6.1 Concept of the Distributed Intrusion Detection Approach

In the previous Chapters of this manuscript, we have assumed that every monitor of our IDS has access to every event from the network traffic it requires in order to evaluate a security pattern. This was possible since a single instance of IDS was running. However, when more detection capacities are required, the number of monitors may increase and they may need to be dispatched across the system to better balance the detection task. Hence, security patterns might be defined over events which are distributed over the system and not accessible for every single monitor, each one having access to only a local subset of the security patterns. In order to correctly evaluate such global security patterns, some monitors have to cooperate in some way. This is the situation that motivated a distributed deployment of our intrusion detection approach. We aim at structuring the monitors in a way that global security patterns can be correctly evaluated in order to detect attacks violating global system requirements. Hence, in such distributed deployments, some attacks would go undetected by local monitors and would only be detected by monitors evaluating global properties. Such attack scenario is detailed in Section 6.5.

Our distributed intrusion detection approach is composed of multiple IDS nodes. As defined in Section 2.3, nodes are used to refer to the processing entities in the different deployments (centralized, decentralized and distributed systems). In order to highlight the difference between a node in the current distributed deployment and the single IDS case presented in the previous Chapters, in the following we call the IDS instances: Intrusion Detection Nodes (IDNs). Each IDN is structurally similar and encompasses identical modules in a layered manner. Their structure is presented in Figure 45.

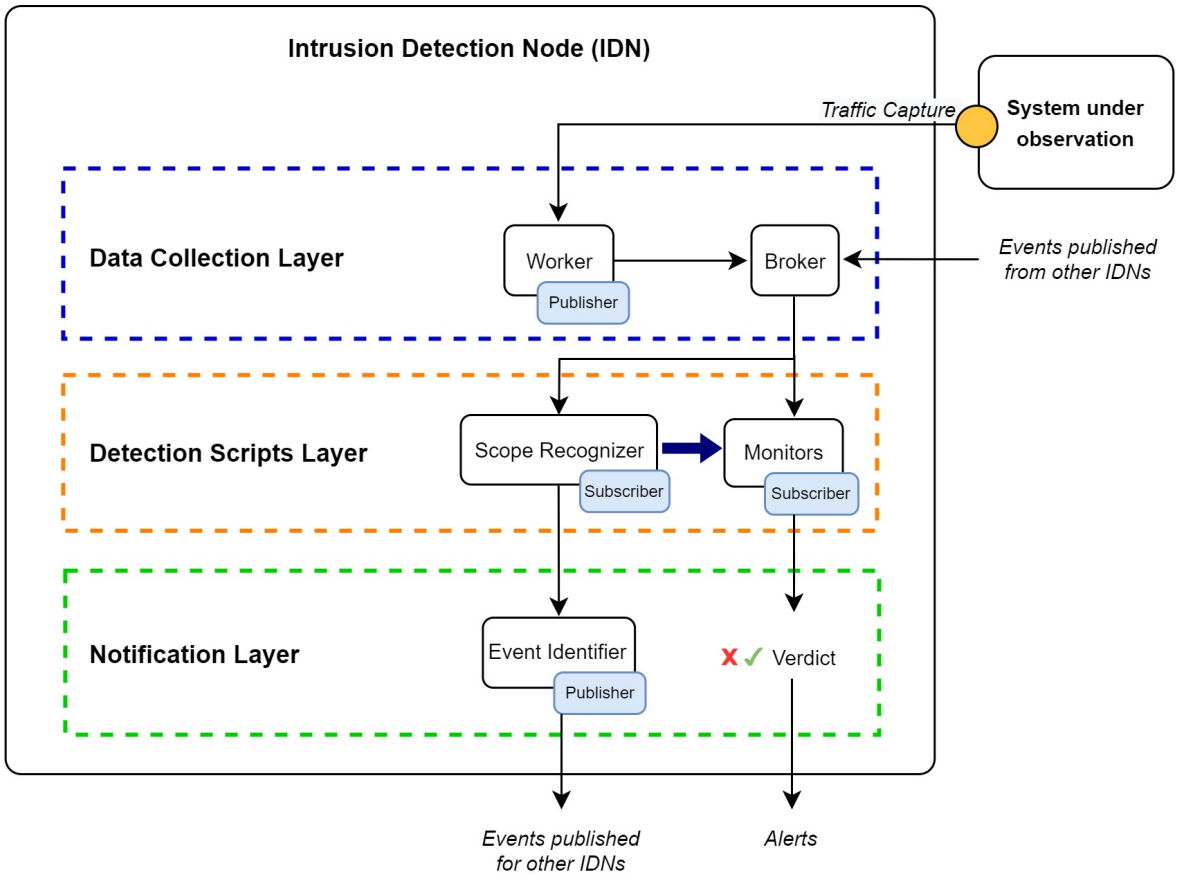

Figure 45: Structure of the layered IDN

At the *data collection layer*, a Broker collects data from two types of publishers: (i) raw

data from a network traffic capture of the system under observation, (ii) events published by other IDNs. Each IDN is tailored to capture the specific network traffic that flows at its vicinity. An IDN can encompass from zero to multiple traffic capture points. The *detection scripts layer* encompasses monitors and the Scope Recognizer. These two modules subscribe to topics of interest on the Broker, for their detection task. The *Notification layer* consists in raising alerts, and publishing events to other IDNs. Alerts are events that violate security policies. Some events which are not alerts are needed in the case of *global security patterns* evaluation. Such global properties include two or more local nodes information.

From the IDS presented in Chapter 4, the only difference in the structural architecture is therefore the ability for IDNs to communicate. Indeed, the previously introduced IDS included a single instance. Thus, it required the ability to raise alerts only. Figure 46 identifies the IDNs layers relatively to the previously introduced framework.

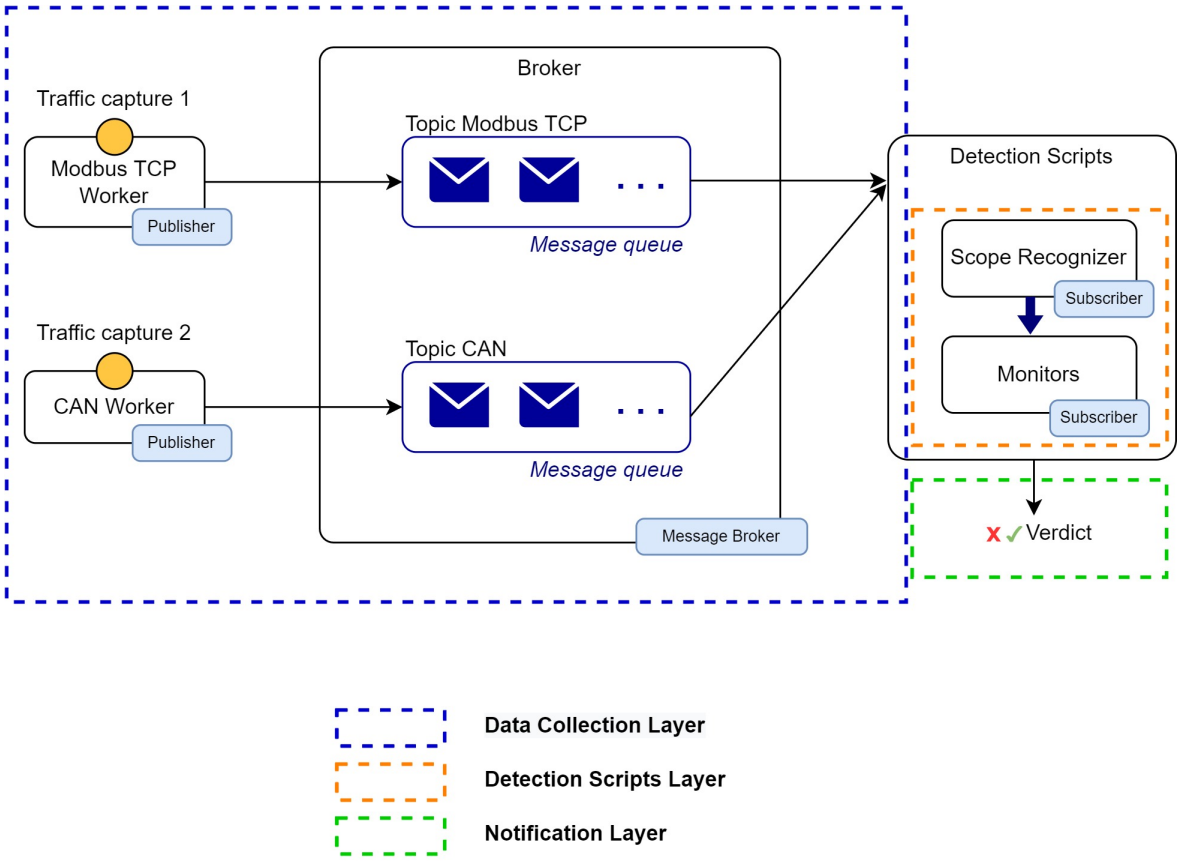

Figure 46: Identification of the layers in the previously introduced framework (see Chapter 4)

Within our framework, there is no central detection unit. Furthermore, there are no hierarchies of IDNs in the sense that they are able to operate independently and raise alerts. However, there is one IDN involved in the monitoring of global security patterns, that subscribe to the events provided by other IDNs. Each IDN is tailored to collect the specific data of its observation domain. This means that each IDN subscribes to the required topics (from the Message Broker) for its application. It also means that the Scope Recognizer and monitors are adapted to where the IDN is deployed. Thus, some IDNs may be positioned at local loop level (as the one in Chapter 5) and monitor fieldbuses events, whereas others may monitor global security patterns and monitor only events provided by other IDNs. If IDNs are structurally identical, they are tailored to their environment and each module has to be adjusted accordingly. However, there is a certain genericity in the modules' codes: (i) Workers, once adapted to a certain network protocol can be reused, (ii) Brokers are identical in each IDN, (iii) Scope Recognizers identify states/events of equipment at their observation domain; therefore, code reuse is possible for the same type of equipment, (iv) Monitors code

can be reused because of the generic specification patterns and (v) Event Identifier is fully dependant on the use case.

An example of a distributed deployment of our framework is represented in Figure 47. An IDN does not necessarily capture network traffic. Some local IDNs can cooperate for the evaluation of some patterns requiring such synchronisation. In the figure, this is the case for the two bottom IDNs. For example, it could be necessary to evaluate if two servo drives (dispatched in the system, in two distinct local loops) are in the same functioning modes.

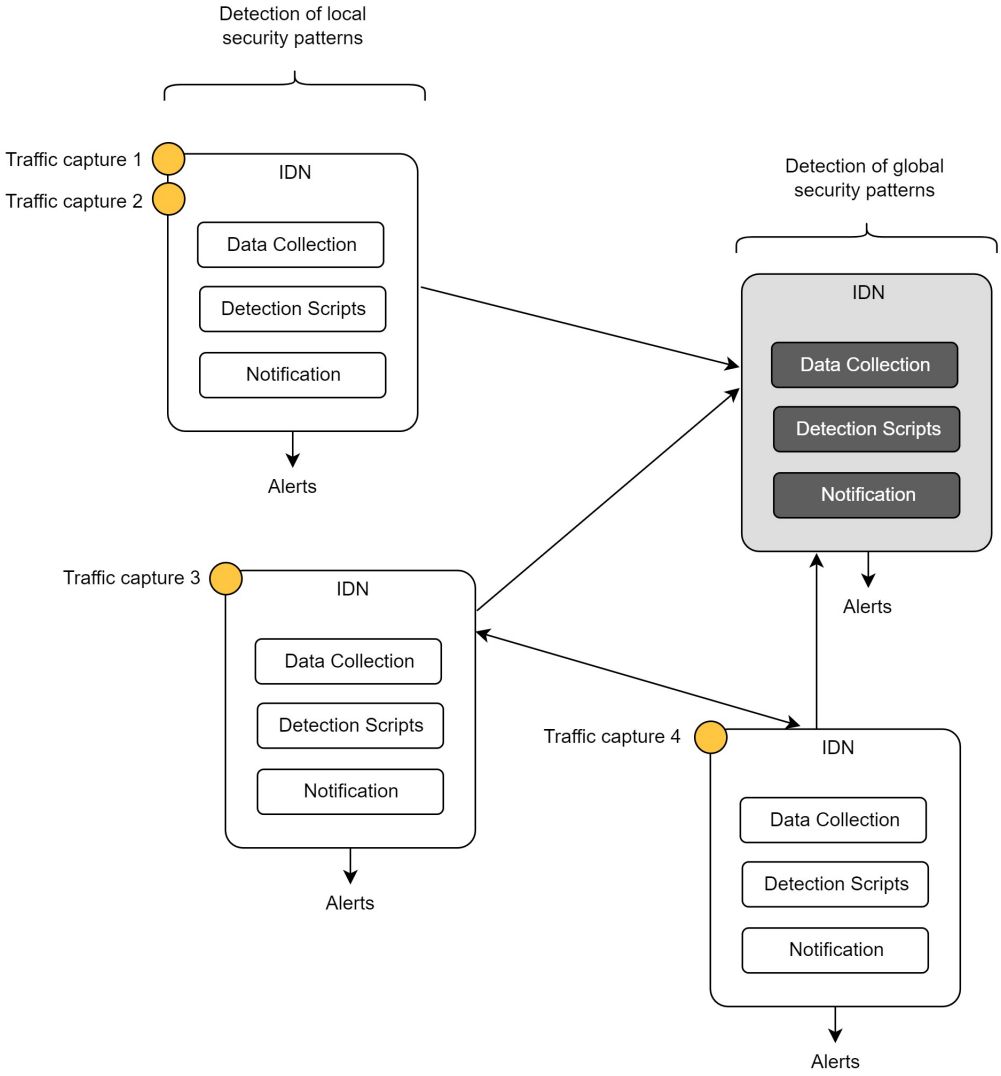

Figure 47: Example of a distributed deployment of several IDNs

6.2 Use case Presentation – Manufacturing Plant

In order to execute experiments on distributed intrusion detection, we define a multiple workstations production line composed of two cartesian robots, one transfer robotic arm (two axis pick and place), a part optical sorting workstation, and a complex conveyor system.

The two real cartesian robots are identical to the one described in Section 5.1. A Factory I/O[1] simulation reproduces the other components of the physical system (manufacturing plant) plus a remote I/O unit that we couple to a real PLC using Modbus TCP/IP protocol.

The assembly line is shown in Figure 48. Between workstations, conveyor belts and presence sensors are used to move the parts. Three types of product exist: metal parts, green parts

[1] https://factoryio.com/

and blue parts. Figure 49 shows the organization of the manufacturing plant from above. The normal flow of operation follows manufacturing routing sheets, displayed in Appendix B for each type of part. The routing sheet is a document that describes the sequence of operations to be executed in the manufacturing process.

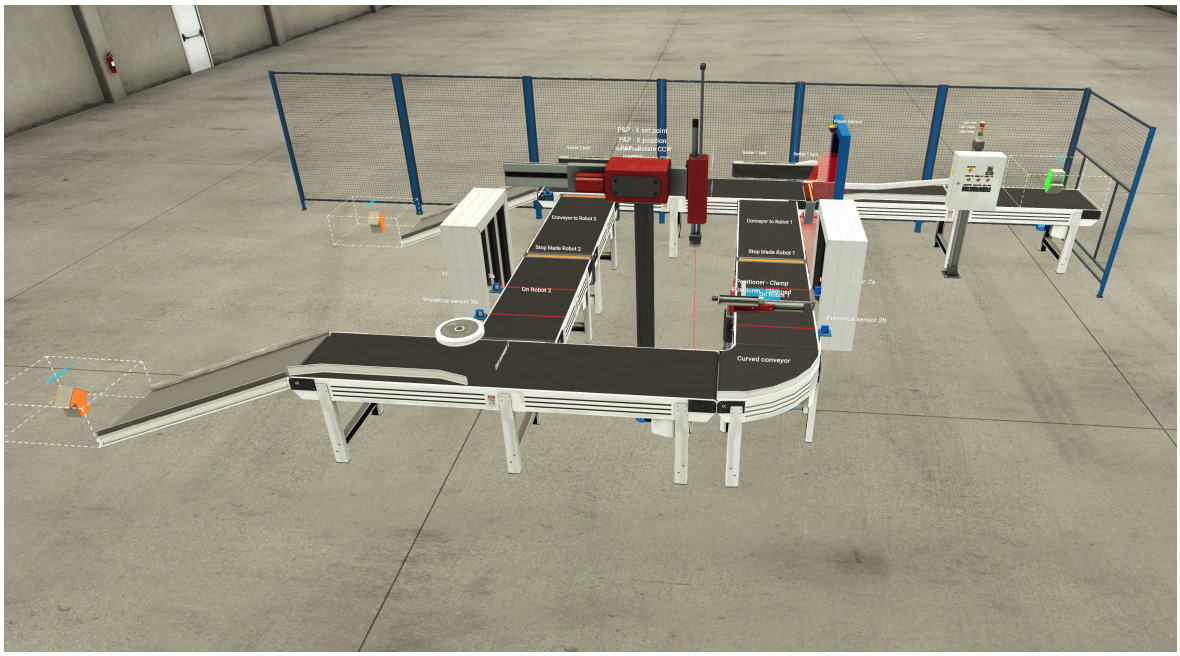

Figure 48: Manufacturing plant overview © All rights reserved

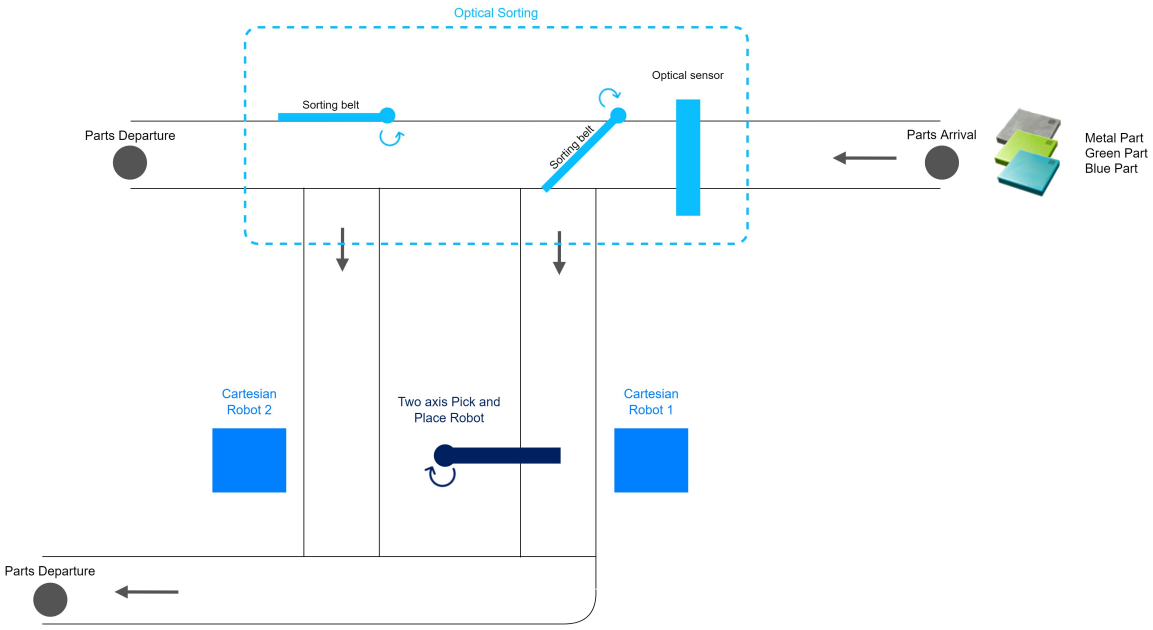

Figure 49: Structure of the manufacturing plant, seen from above

This use case is a complex industrial system with various subsystems; its overall network architecture is depicted in Figure 50. On this Figure, for clarity matters, only one cartesian robot is represented. In the remainder, the examples will also lie on a single cartesian robot. There is a PLC running the control program of the simulated part of the factory (optical sorting machine, pick and place robot and various sensors and actuators like presence sensors and conveyor belts). All the devices are connected to the same LAN.

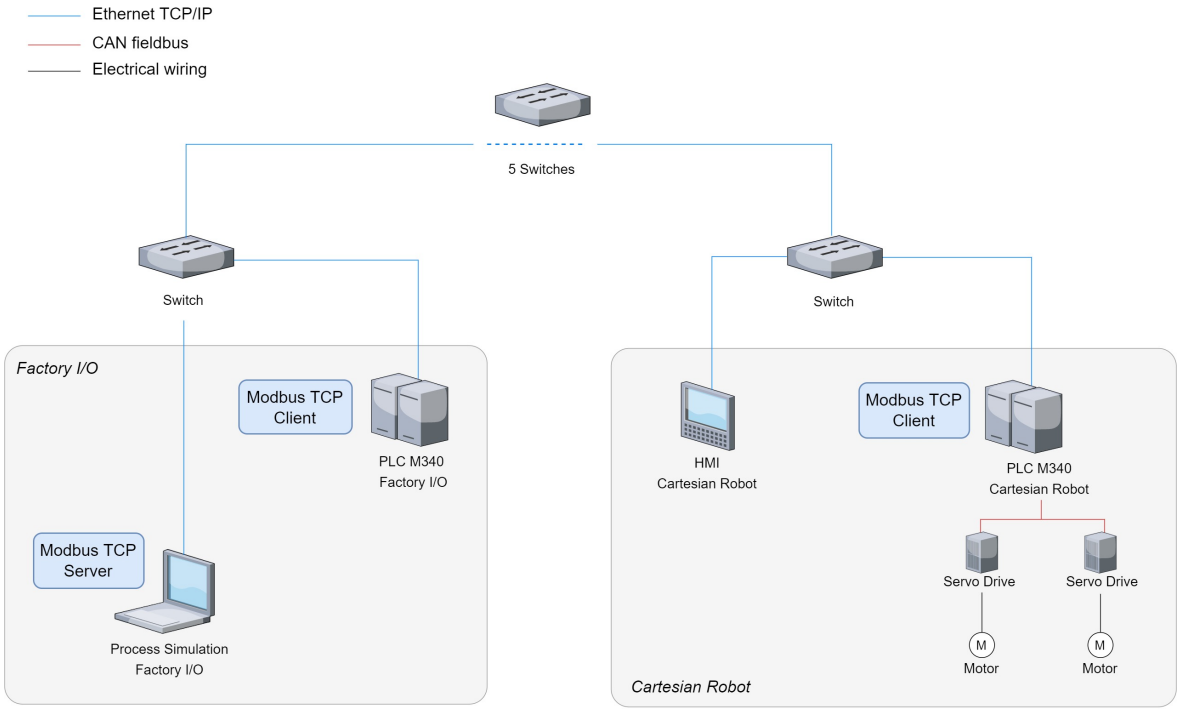

Figure 50: Manufacturing plant network architecture

6.3 Security Patterns Synthesis Process

In our distributed deployment, in addition to local security patterns, there are global security patterns to evaluate (see Section 2.3.3 for a definition and example).

For each IDN, the deployment of the whole detection framework (security properties extraction, security patterns synthesis and runtime monitoring) follows the methodology explained in Chapter 4 and illustrated in Chapter 5. However, there are also global security properties. The evaluation of these global security properties relies on the events exchange between IDNs. We are going to detail next the security patterns synthesis process for global security patterns.

6.3.1 Global Security Patterns

Best practices require that implementation documents should be available to system users. Such documents would be useful to obtain system's global requirements. In real life, they are rarely available (owned by third parties or no back-up available). However, within the manufacturing industry, the routing sheets of the parts provide specifications for the system. From these documents, we can for example extract sequences of operations for the production line.

If no documentation can be used to obtain system's global specifications, expert knowledge is required and a human specialist has to list the system's global requirements which is a time-consuming and error-prone task. This is one major difference to the specification of local requirements (used in the previous Chapters), where the common characteristics of safe behaviors could be extracted from standards directly. This was possible because controllers and networks in ICSs conform with international standards. Unfortunately, there are no standard used in a consensual manner for high level system requirements. Once the specifications are listed, network traffic observations are required to refine the properties, as sometimes they are not entirely specified from the previous step. For example, ordering in events or timing information if needed may be retrieved from traffic observation and analysis.

In our use case (and in any manufacturing plant), the production flow is determined by the routing sheets: each product is going to successively be processed by some of the workstations. For instance, as shown in the routing sheet, Table 14 (in the Appendix), the blue part is cut by cartesian robot 1, then moved by the two axis pick and place robot, cut by cartesian robot 2, and finally removed from the assembly line. One system safety requirement would be, for example, for each manufactured part, to respect the order of operations conducted by workstations. For instance, the two axis pick and place robot should move the blue part only after cartesian robot 1 has finished its operation on the part. In terms of security patterns, it means that: (i) two axis pick and place robot being busy responds to cartesian robot 1 not being busy; (ii) cartesian robot 1 not being busy responds to cartesian robot 1 being busy; (iii) cartesian robot 1 being busy precedes cartesian robot 1 not being busy; and (iv) cartesian robot 1 not being busy precedes two axis pick and place robot being busy. Hence, the corresponding global security patterns are two *Response* ((i) and (ii)) and two *Precedence* ((iii) and (iv)) patterns. From network traffic observations, information are retrieved concerning the meaning of such information at network level. Every global security pattern is evaluated with a global scope.

Optionally, one can also extract time information from network traffic observations. This time information can concern the processing time of a workstation or the elapsed time between two workstations successions. This permits to take into account attacks that aim to slow down the production process.

Hence, the four global security patterns can be expressed as the following LTL formulas:

$$\Box(cartesian\_robot\_1_{busy} \rightarrow \Diamond cartesian\_robot\_1_{not\_busy}) \tag{4}$$

$$\Box(cartesian\_robot\_1_{not\_busy} \rightarrow \Diamond pick\_and\_place\_robot_{busy}) \tag{5}$$

$$(\neg pick\_and\_place\_robot_{busy}\ \mathcal{W}\ cartesian\_robot\_1_{not\_busy}) \tag{6}$$

$$(\neg cartesian\_robot\_1_{not\_busy}\ \mathcal{W}\ cartesian\_robot\_1_{busy}) \tag{7}$$

The monitoring of the four global security patterns requires information from two distinct workstations: the cartesian robot 1 and the two axis pick and place robot. Hence, it is impossible to obtain a verdict locally. Each local IDN has to communicate its local information to the IDN evaluating the corresponding global security patterns.

6.3.2 IDN for Global Properties

In order to evaluate global security properties, we use a specific IDN that conducts (alone) all the monitoring processes while receiving local events from other IDNs. This is the *orchestration* setting presented in [34] from the field of runtime verification. This global IDN does not capture any traffic. In order to evaluate the global security patterns (4), (5), (6) and (7), local IDNs have to publish the following events: $cartesian\_robot\_1_{busy}$, $cartesian\_robot\_1_{not\_busy}$ and $pick\_and\_place\_robot_{busy}$.

However, these events are abstractions of physical process states seen in the network traffic. Reciprocally, physical process states seen in the network traffic are concretions of these high level events. While abstraction is the fact of hiding complexity in order to make something simpler to understand or to use, concretion is the opposite: it is the fact of dealing with specific details. Multiple ways exist to concrete (i.e. to go from abstractions to concretions) an event: for instance $cartesian\_robot\_1_{busy}$ abstraction can be obtained both via fieldbus or Modbus TCP by checking the state of different process variables.

The three abstract events:

$$cartesian\_robot\_1_{busy},\ cartesian\_robot\_1_{not\_busy} \text{ and } pick\_and\_place\_robot_{busy}$$

can be obtained using the Scope Recognizer. The Scope Recognizer encompasses a FSM with the functioning modes of the servo drives, including "no movement". If both servo drives are in the "no movement" state, then

$$cartesian\_robot\_1_{busy} = 1 \text{ and } cartesian\_robot\_1_{not\_busy} = 0.$$

A similar process is conducted for every abstract event. Then, the role of the Event Identifier (see Figure 45) is to publish such abstract events.

6.4 Runtime Monitoring

6.4.1 Data Capture

In this use case, the detection task relies on three IDNs. IDN 1 and IDN 2 are located at subsystems and they evaluate local security patterns in order to detect abnormal behaviors of subsystems. IDN 3 evaluates global security patterns and receives events from IDN 1 and IDN 2. Figure 51 shows where capture points are located. The capture points and the IDN deployed at the cartesian robot workstation are identical to the one described in Chapter 5. On the practical aspect, each IDN is deployed on a computer and passively observes network traffic. The communication between IDNs is performed on a dedicated network.

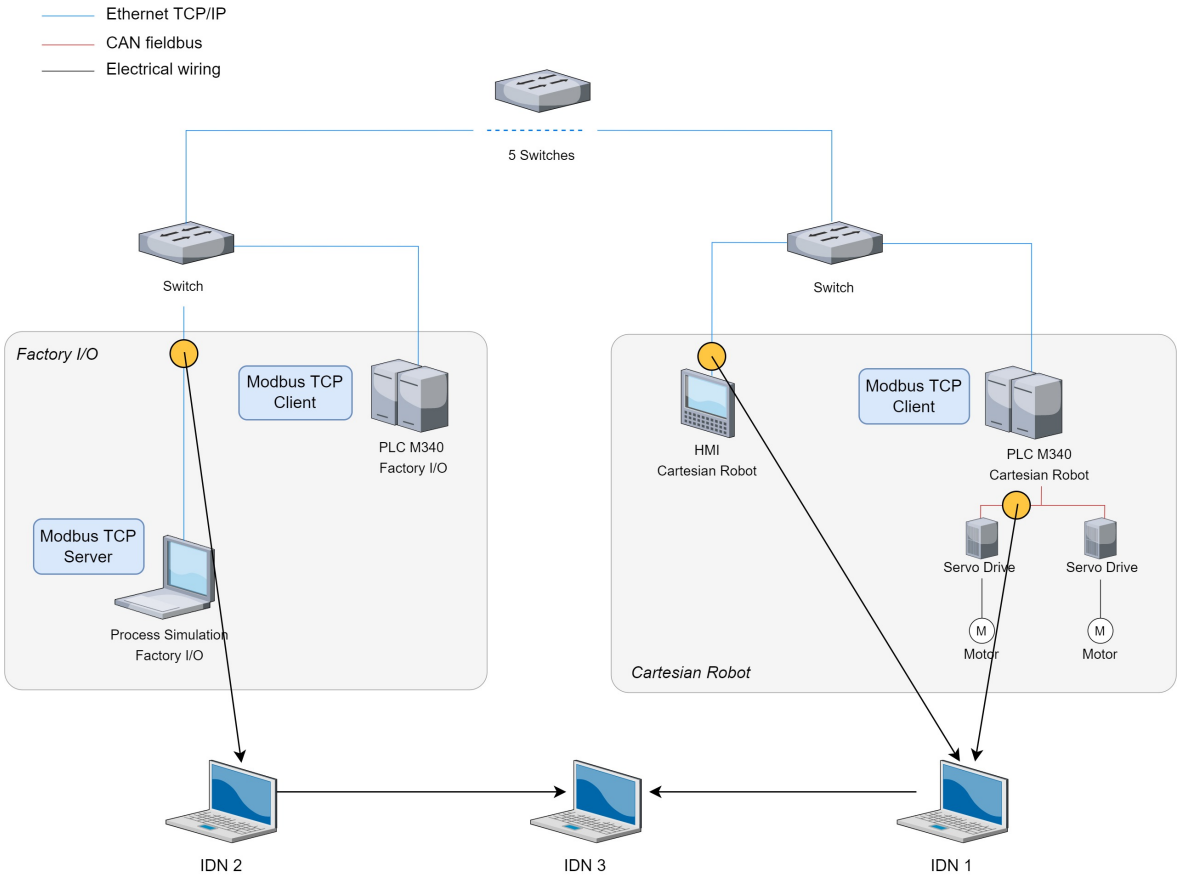

Figure 51: Traffic capture points of the use case for a distributed deployment

6.4.2 Scope Recognizer and Monitors

The routing sheets are used to define system's states for the **Scope Recognizer**. The Scope Recognizer identifies the workstations changes for each part currently on the production

line. Therefore, there would be as many FSMs as the number of parts, and the states of each FSM would indicate workstations.

Concerning the **monitors**, the experiment is performed with 4 monitors (one for each pattern) evaluating global security patterns, in addition to the monitors evaluating local security patterns.

6.4.3 Software Deployment Engineering

As for the previous experiment, we rely on the open source NIDS Zeek for the deployment of our IDNs. In order to easily operate and manage Zeek accross the multiple IDNs, we use ZeekControl[2]. ZeekControl is an interactive shell that allows to manage multiple Zeek installations in a traffic-monitoring cluster. A *Zeek Cluster* is a set of computers (the IDNs in our use case) jointly analysing traffic in a coordinated manner. Each IDN is a computer of the Zeek Cluster; these computers are not part of the industrial system under observation. The coordinating unit is called the Manager, which is a process able to start/stop/get states of the rest of the computers in the Cluster. Figure 52 gives a representation of this process.

In our use case, the Manager is on IDN 3. We specified our deployment architecture on configuration files intended for this purpose (containing IP addresses of the remote machines for instance). Based on these configuration files, the manager launches workers on IDN 1 and IDN 2. The manager also deploys custom Zeek scripts, which are related to the functioning of workers on each IDN. From a practical aspect, all machines in the Cluster (i.e. all IDNs) have to run the same version of their operating system and must be configured to run Secure Shell (SSH) in a secure way.

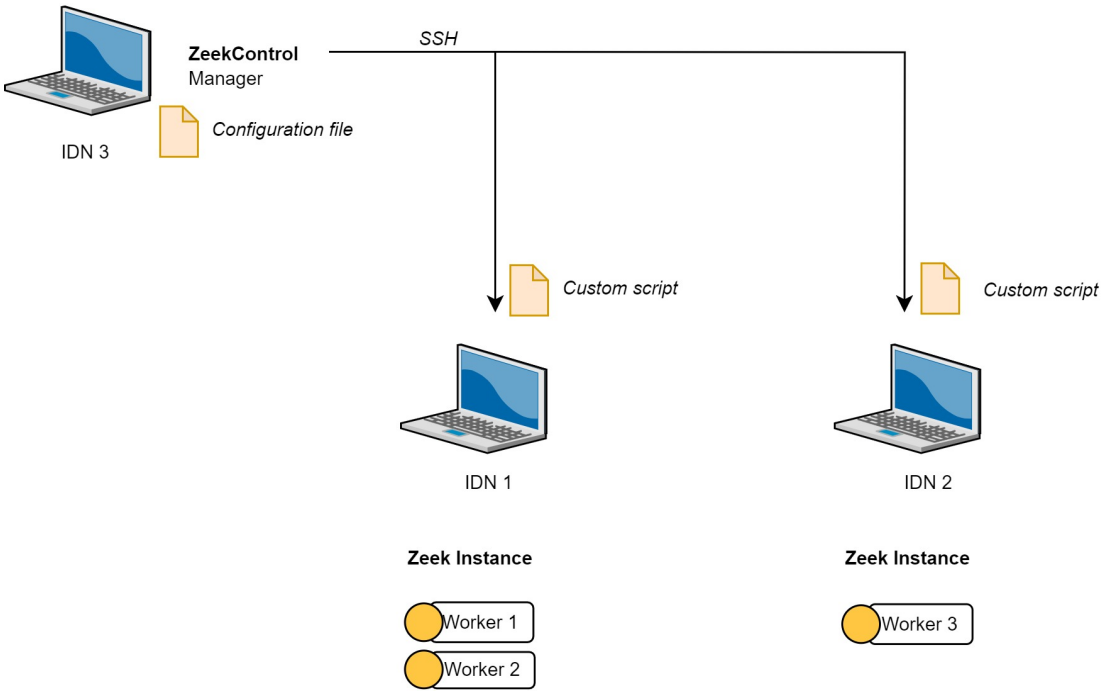

Figure 52: Managing Zeek over the IDNs using ZeekControl

The communication security is provided by Zeek secure communication process. Furthermore, IDNs communicate on a dedicated network, completely separated from the ICS. Thus, they have no impact on the industrial process. Concerning time synchronization between IDNs, we use the Simple Network Time Protocol (SNTP)[3] of the industrial network. Even though the IDNs have their own communication network, they need the same synchronisation

[2] https://github.com/zeek/zeekctl
[3] https://datatracker.ietf.org/doc/html/rfc5905

precision than the ICS they intend to monitor. Hence, one limitation of this configuration is the vulnerability induced by the use of the ICS SNTP, needeed for time synchronisation of the IDNs with the industrial network.

6.5 Attack Scenario

This Section develops an attack scenario that would go undetected locally, in order to motivate the monitoring of global security patterns within our distributed detection framework.

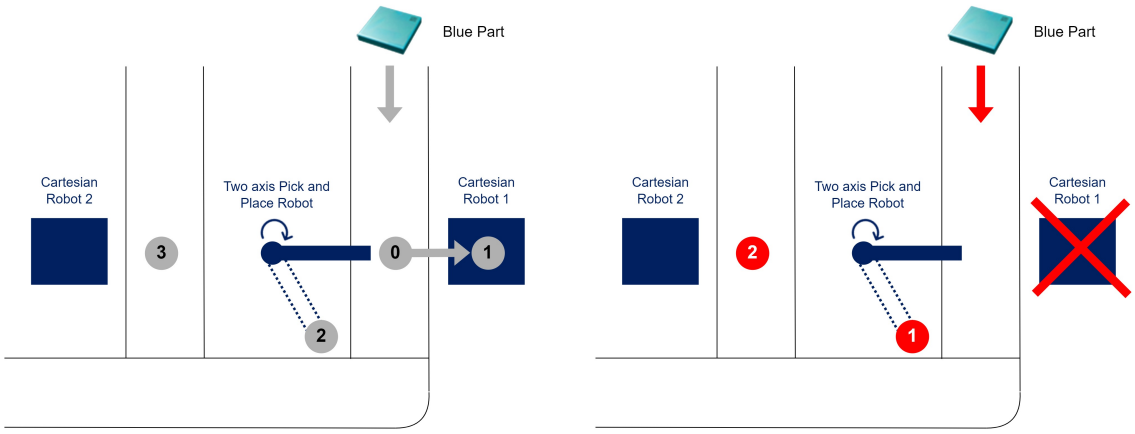

Figure 53: Illustration of the normal scenario (left) and attack scenario (right)

The normal scenario is represented in Figure 53, on the left side. We consider the routing sheet for the blue part. The parts arrive from a conveyor and stop in front of cartesian robot 1 (position 0). The cartesian robot 1 processes the part (position 1) and put it back to the conveyor. Then, the pick and place robot moves the part (position 2) to another assembly line (position 3).

The synchronisation between workstations is implemented via network variables:

$$Part\_Ready\_for\_Robot1 \text{ and } Part\_Out\_of\_Robot1.$$

$Part\_Ready\_for\_Robot1$ is written by the PLC containing the codes of the simulated part of the factory. $Part\_Ready\_for\_Robot1$ switches to 1 when a part is in front of the Cartesian Robot 1 (i.e. when the part is detected by presence sensors, at position 0). The network variable $Part\_Ready\_for\_Robot1$ is periodically read by the cartesian robot's PLC at each PLC cycle. The cartesian robot's PLC starts processing the part, whenever $Part\_Ready\_for\_Robot1 == 1$. Similarly, $Part\_Out\_of\_Robot1$ is written by the PLC of the cartesian robot at the end of the blue part's processing. It is periodically read by the PLC containing the codes of the simulated part of the factory, at each PLC cycle. The pick and place robot's PLC starts moving the part, whenever $Part\_Out\_Of\_Robot1 == 1$. The pseudo code for the cartesian robot's PLC is shown in Algorithm 2.

We deployed an attack whose aim is to bypass the operation of Cartesian Robot 1 on the part. The attack scenario is represented in Figure 53, on the right side. The attack forces $Part\_Ready\_for\_Robot1$ to zero, permanently, on the PLC containing the codes for Factory I/O. This forbids the cartesian robots to start processing a part, even if one is available. Then, each time a blue part arrives in front of Cartesian Robot 1, it forces $Part\_Out\_of\_Robot1$ to 1 on the PLC of the cartesian robot. The workstation synchronisation via network variables is illustrated in Figure 54 for the normal scenario (left) and the attack scenario (right). The attack scenario corresponds to a *distributed attack* as it involves various targeted PLCs, from different workstations.

Algorithm 2: Pseudo code for the cartesian robot's PLC

Data: *Part_Ready_for_Robot*1: shared network variable
start: condition to start processing the part
PLC_FactoryIO: PLC containing codes of the simulated part of the factory
Result: *Part_Out_of_Robot*1: shared network variable

start ← *PLC_FactoryIO.read*(Part_Ready_for_Robot1) ; `// read cyclically`

if *start* == 1 **then**
    process_blue_part;
    Part_Out_of_Robot1 ← 1

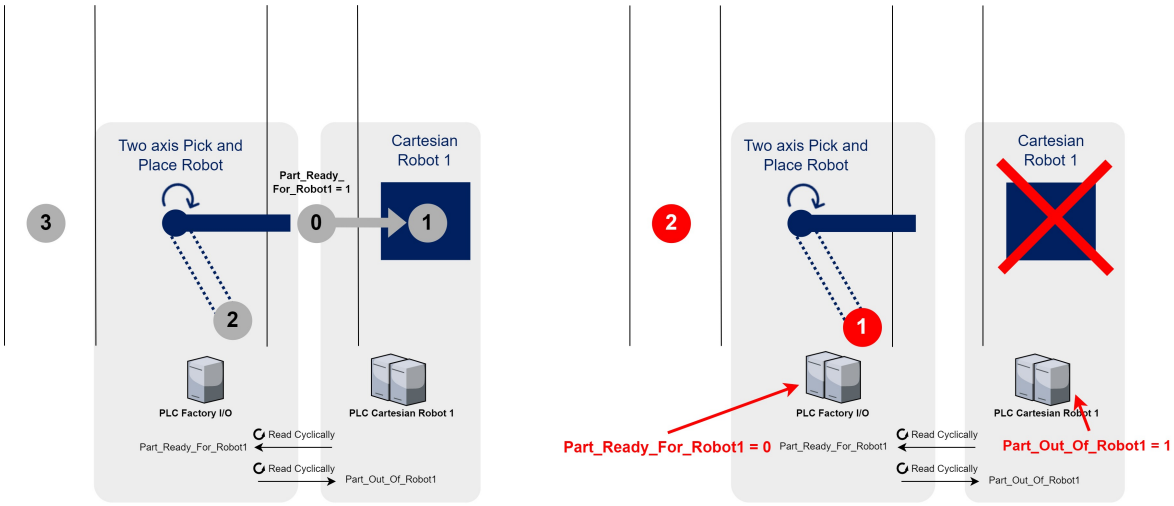

Figure 54: Workstation synchronisation for the normal scenario (left) and attack scenario (right)

The consequences of such an attack, can be visualised on the physical process. In our use case, when an attack is launched, we visually see blue parts on the assembly line go straight from the conveyor to the pick and place robot. This results in a damaged part as the part is missing a workstation operation. Visually, such attacks targeting the sequence of workstation operations can lead to big confusion on the assembly line. Figure 55 shows an important number of parts that were not processed by Cartesian Robot 2 as a result of one attack on the sequences of events. All the parts piled up in front of Cartesian Robot 2 before leading to the whole production to stop.

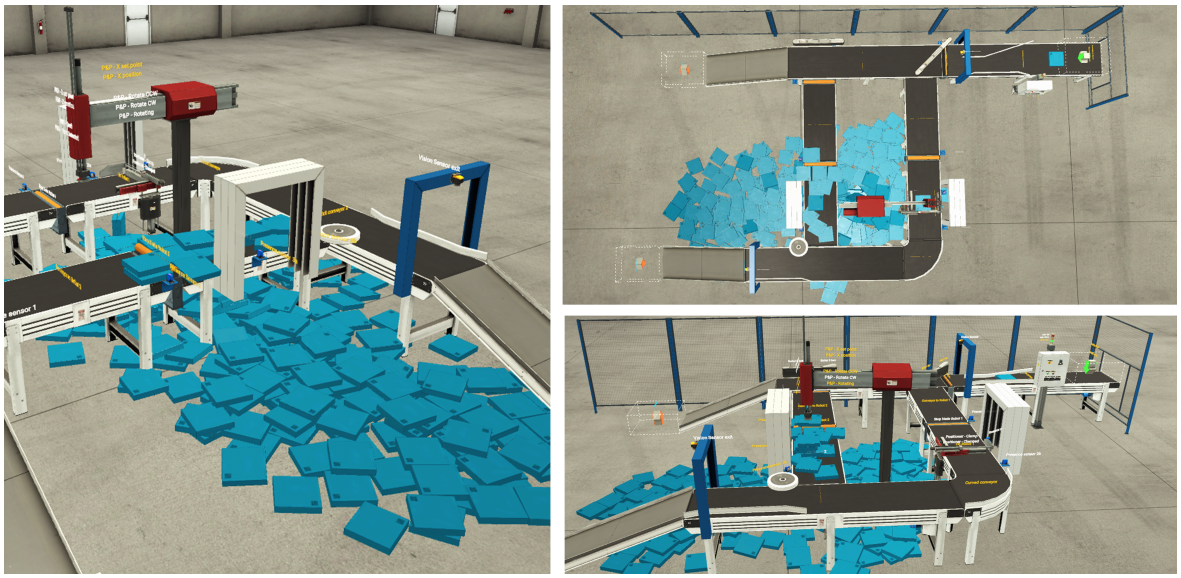

Figure 55: Visual effect of an attack disrupting the sequence of workstations' operations © All rights reserved

6.6 Evaluation

Our approach being process aware, traditional network-security performance measures are not sufficient. This is the reason why we consider operational and time-based metrics for evaluating our distributed detection framework.

Furthermore, we do not focus on monitor-dependent measurements (such as the time it takes to update monitors, the maximal number of monitors or the actual overhead of monitors) since these results are already investigated in Chapter 5. We rather focus on metrics related to the monitoring of global security properties:

- **Detection capabilities.** This part investigates which type of attacks are detected by the approach, and time performance evaluation is conducted.
- **Scalability.** This part evaluates the scalability of the approach.
- **Extensibility.** This part discusses the generalisation of the approach and its usage on other cases.

6.6.1 Detection Capabilities

As previously presented, our approach relies on 4 monitors for the evaluation of global security patterns in addition to monitors for local security patterns. For the evaluation, every monitors are activated, including the monitors evaluating local security patterns. The latter

are the one described in the previous Chapters and deployed on the cartesian robot (33 monitors).

Attack implementation and effect on the physical process.

The attack scenario described in Section 6.5 is launched through a Python script on Modbus TCP network using a Modbus TCP client tool. It was launched 361 times.

We also implemented a variation of the previously described attack by launching it from fieldbus level. The first step of the attack remains identical: $Part\_Ready\_for\_Robot1$ is forced to zero permanently, on the PLC containing the codes for Factory I/O. This step is done through Modbus TCP network. Then, instead of forcing $Part\_Out\_of\_Robot1$ to 1 on the PLC of the cartesian robot when a part is available, we trigger a command that has the same effect at fieldbus level. On the cartesian robot PLC, if an attacker reverse-engineers the programs for the processing cycle of the blue part, the attacker can deduce which command validates the end of the processing cycle in order to automatically switch $Part\_Out\_of\_Robot1$ to 1. This way of implementing the attack is dependent on the way PLCs are programmed, and the use case. However, it highlights the fact that our distributed detection framework can detect violation of global safety policies, triggered from a fieldbus.

Attack detection and response time.

We are able to detect global security patterns disrupting sequences of events, or optionally impacting timing of events (if specifications on timings are available in the use case). Furthermore, we are able to detect attacks launched from fieldbus level, triggering global security patterns.

The results in Table 10 show the characteristics of the conducted attacks. The attack disrupts the sequence of operations in the shopfloor. For 361 repetitions, the Average Response Time (AvRT) is 91.0 ms, the Worst Case Response Time (WCRT) is 96.2 ms and the standard deviation (σ) is 1.58 ms. The experimentation follows a unimodal, positively skewed distribution with a mode at 90.5 ms, represented in Figure 56. The AvRT being close to 90 ms is a satisfactory result. It can be explained by the fact that from the time the attack is launched to the time it is detected, variables are read from both the PLC containing codes of the simulated physical process and the PLC of the cartesian robot. Thus, at worse, one entire cycle for each PLC occurs. The cycle time of PLC containing codes of the simulated physical process is 80 ms, whereas the cycle time for the PLC of the cartesian robot is 10 ms.

Table 10: Implemented attacks and their effect on the system

Category of attack	# Attacks	Entry point	Physical effect	WCRT (ms)	AvRT (ms)	σ (ms)
Disrupt sequence and timing of operations in the shopfloor	2	Modbus RTU	Damage parts	96.2	91.0	1.58

Our specification-based approach permits to define correct behavior of the system in order to detect any deviation from it. Therefore, an attacker can avoid being detected if he does not trigger a misbehavior (i.e. misbehaving state of the IDSs). We argue that in this case, no harm will be caused on the ICS nor its environment since no security properties are violated.

6.6.2 Scalability

The distributed deployment is by nature a solution to make our framework scalable. In our contribution, we did not face any network-induced delays nor data packet disorder.

We evaluated the bandwidth of the communication channel between IDNs. The bandwidth according to IEC 62443 [89] standard is the capacity of a communication channel to pass data through the channel in a given amount of time. Bandwidth is usually expressed in bits per second. We evaluated the time it takes to pass on an event from an IDN (Publisher) to another (Subscriber). The results are shown in Table 11, for 100 executions. The average

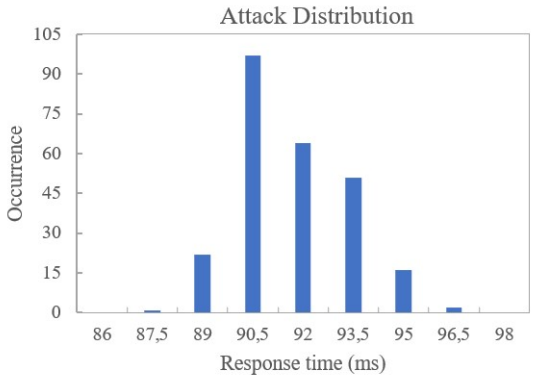

Figure 56: Distribution of the response time for the detection of the attacks

transmission delays are way inferior to the sampling period of the events of interest. Indeed, the events published by Event Identifiers are updated at every cycle of PLCs (cycle times of 10 and 80 ms).

Publisher	Subscriber	Average transmission delay (ms)
IDN 1	IDN 3	0.0086
IDN 2	IDN 3	0.0068

Table 11: Transmission time of an event between IDNs

Due to the publisher/subscriber used in our framework, we have a limited latency in the communication of events from the Event Identifier to the Broker of a given IDN. Hence, the messages are delivered in a push-based manner, which means that the publishers initiate requests to subscribers to deliver the message whenever they have a message to deliver. It prevents monitors to periodically check for new information or updates. This promotes faster response time and reduces the delivery latency, which is an important asset in systems meeting real-time constraints.

Once deployed, industrial systems are unlikely to be heavily modified. However, they can evolve with sparse addition of components or operating requirement changes. The architecture of our distributed intrusion detection framework allows such evolution to happen. Indeed, it is a flexible architecture since IDNs can operate independently. For instance, adding one IDN monitoring local security patterns will not have any impact on other IDNs. However, adding new global security patterns is a more timely task as the choice of events to propagate to global detection nodes will have to be discussed together.

6.6.3 Extensibility

The generalisation of the approach and its usage to other use cases has been already discussed for local IDNs. Concerning IDNs monitoring global security properties, the main challenge is on obtaining the specifications. This aspect depends on the availability of system specifications. Our use case relied on the routing sheets for both the specifications extraction and the Scope Recognizer. One limitation of the methodology is that the specification extraction for global security properties is use case dependant.

Conclusion

In this Chapter, we have presented a distributed deployment for our intrusion detection approach. The intrusion detection task is performed by several IDNs presenting a three layers architecture: data collection layer, detection scripts layer and notification layer. Each IDN

is structurally identical. Some IDNs evaluate local security patterns and one IDN evaluates global security patterns. IDNs monitoring local security patterns are able to raise alerts and publish local events required for the evaluation of global security patterns. The IDN monitoring global security patterns subscribe to local events in order to evaluate properties that could not be evaluated locally solely.

Regarding our use case, we presented a distributed attack scenario, disrupting the workstations' sequence of operations in a production line. We evaluated our distributed approach and showed its detection capabilities regarding global security patterns. It has good response time results for detection, that is under 100 ms for global security properties. Furthermore, we show that our approach is scalable and extensible.

Conclusion and Perspectives

In this thesis, we have presented a specification-based, process aware, intrusion detection system, monitoring security properties on the network traffic for complex hierarchical industrial control systems presenting hybrid dynamics. We first detailed the methodology for our intrusion detection approach on a simple hierarchical ICS testbed. Secondly, we investigated a distributed deployment for our IDS approach for larger testbeds.

The motivation for this contribution arises from the context and the specific properties that define ICSs. This is the topic of Chapter 1. The question of ICS cybersecurity is at most a few decade old and is characterized by the integration of IT within OT networks. The cybersecurity landscape of ICSs has expanded to include vulnerabilities that were not previously considered. Securing ICSs is now a major concern, taken very seriously by organizations and governments. However, ICSs have to deal with strict time constraints especially in their lower levels, they encompass heterogeneous networks and resource-constrained equipment. Due to these particular features, we have identified behavior-based intrusion detection as a promising security control since it is able to passively monitor network traffic and it can be positioned at multiple network levels. Another important characteristic of ICSs is the presence of a physical process. Recent years have witnessed the appearance of sophisticated attacks aiming at disrupting the physical process of ICSs. We call these attacks *process aware* and we address their detection in our contributions.

Through an extensive review of the literature concerning behavior-based intrusion detection, we have discussed several approaches. This is the topic of Chapter 2. We have identified three main categories relatively to which part of the ICS is considered to design the model: communications between equipment, task and resources of equipment, control data and control logic. This last category appeared to be the most adapted to answer the challenge of detecting process aware attacks, since it includes knowledge of the physical process when designing intrusion detection models for ICSs. Even though this category of approach seems promising, they present several difficulties that we identified in order to address them. First, many approaches have a limited comprehension of the physical process. We highlight the granularity in physical process knowledge and the importance of considering control data and control logic but also functioning modes of equipment and operational contexts. Furthermore, through this review of work, we highlighted the difficulty to collect data at local loop level, and the fact that very few works have put it into practice. Second, many approaches rely on a costly construction of their detection model. Many approaches rely on human expert knowledge and they present manually derived models. Third, few approaches take into account the notion of *time*, and characterize the full temporal dynamics of ICSs. Fourth, few approaches have evaluated their scalability by adopting a distributed deployment. Centralized intrusion detection may suffer from limited detection possibilities when deployed on larger systems.

Throughout this work, we aimed at answering the previously identified difficulties. Our contributions are detailed in Chapters 3, 4, 5, and 6. Our contributions present the following original points:

- The development and deployment of a specification-based intrusion detection approach relying on the monitoring of security properties against the system execution. Our

approach encompasses high physical process knowledge including control data, control logic, as well as functioning states and modes of components. Our approach is deployed on the communication network between the distributed control and the local loops and between the supervisory control and the distributed control of the system. Therefore, we are able to perform detection at the lowest network level, i.e. fieldbus level, which is the closest to the physical process.

- The security properties for the monitoring are systematically retrieved from international and industry standards, and may be refined with network traffic observation. The security properties need to be retrieved from the standards only once for each type of device as they are common among all the manufacturers compliant with the standard.

- We use expressive formalism from TL family of specification language in order to evaluate our security properties and tackle the hybrid dynamics of ICSs. Using results from runtime verification, we were able to develop a lightweight and low complexity approach.

- We investigated a distributed deployment which reduces dependence on a single detection unit and helps to balance the intrusion detection task across multiple detection nodes. Each detection node is structurally identical. This structure permits to evaluate local security properties at some nodes, but also global security properties at other nodes. Hence, local information affecting global security properties are propagated accordingly.

The above propositions are implemented on two ICS testbeds: G-ICS testbed and Naval Group testbed. These testbeds provide realistic and safe environment to evaluate our contributions. The results are promising and show that the proposed approach has good detection capacities, is scalable and extensible. This work could be expanded upon in the future, and we present research directions below.

Perspectives

We now discuss the possible extensions of our contributions, in the short, medium and long term. In the short term, perspectives are based on the application and further development of our contributions:

- **Creating a tool for the monitor synthesis.** In Section 4.5, we presented the monitor synthesis used in our approach. In our work, an automata-based approach was applied for Dwyer and Maler patterns, whereas a variant of rewriting was used for Maler patterns. It could be interesting to investigate the choice in the decision procedure for the monitors synthesis by conducting a comparative study on the numerous library and tools that already exist (e.g. Reelay[4] for past-MTL and LTL, Breach[5] for offline STL monitoring, TeSSLa[6] which brings its own temporal specification language, etc.). To our knowledge, no library nor tool tackles all three specification formalisms used in our work: STL, MTL and STL. Hence, one direct application of our work would be to create a library (preferably using a programming language compliant with real-time constraints such as C or C++) that would generalize and optimize the monitor synthesis for the TL formalisms used in our work.

- **Adding quantitative estimation of the deviation from normal behavior.** It would be beneficial for a human operator to have a quantitative estimation of the deviation from the system's normal behavior instead of a binary verdict when a security

[4] https://github.com/doganulus/reelay
[5] https://github.com/decyphir/breach
[6] https://tessla.io/documentation/

property is violated. As mentioned in the state of the art Chapter 2, such kind of quantitative metric has already been studied in the literature, it is the *robustness degree* [51]. It allows to add a degree of satisfaction/violation. This function of time computes the distance to violation of a given STL formula. For instance, if we take the signal predicate $x > c$, the robustness gives the relative position of x to c instead of only indicating whether x is above or below the threshold. While it is particularly well adapted for STL, it could be interesting to derive a similar indicator for LTL and MTL. For this research track, it is worth to consider the notion of distance from the works in [26] and [142].

- **Extending the framework to other fieldbus protocols.** So far, the implementation tackles CAN and Modbus Remote Terminal Unit traffic capture. It could be interesting to extend the detection capacities to other common industrial fieldbuses protocols such as Profibus[7], EtherCAT[8], ControlNet[9], etc.

- **Implementing the distributed deployment on a large scale physical testbed.** In order to be fully realistic and avoid any loss of data fidelity, it would be necessary to evaluate the distributed deployment of our approach on a 100% physical testbed instead of having the physical process simulated.

In the medium term, perspectives could be the following research topics:

- **Automating the security patterns synthesis.** A future extension of our work concerns a complete automation of the security patterns synthesis process. In this manuscript, this latter is systematic, since we rely on predefined patterns and standards operation modes of components. Full automation is not yet possible as standard specifications are generally not provided in a suitable format (like .xml), although some notable exceptions exist like the IEC 61850 standard [87]. It would be interesting to investigate the fields of ML, and more specifically Generative Artificial Intelligence, in order to automatically generate security patterns from standards through adapted prompt engineering.

- **Deploying alert correlation.** It would be necessary to deploy a supplementary module to our contribution that would conduct alert correlation. This could, on the one hand, solve the alert overlapping issue that is a limitation of our work discussed in Chapter 5. Currently, a single attack may violate several security properties, and, potentially, raise a substantial number of alerts, which is challenging to analyze for a human operator. Furthermore, the distributed deployment adds even more instances of IDSs with even more possible alerts. On the other hand, developing an alert correlation module would permit to reconstruct attack scenarios. Alert correlation is a topic of interest in the literature [40] and should be adapted to our contribution. It would be highly beneficial for such alert correlation module to be supplied with our process aware alerts.

- **Monitoring global security properties.** In our distributed deployment, presented in Chapter 6, we detail the evaluation of global security properties. In order to evaluate such global properties, we use a global ICS instance that conducts all the monitoring process while receiving local events from other ICS instances. However, many other techniques exist, from the field of runtime verification, to manage the distributed detection framework [34]. Research would be necessary to decide on the most efficient way to evaluate such properties in order to optimize the detection response time and minimize network delays.

[7] https://www.profibus.com/
[8] https://www.ethercat.org/
[9] https://www.odva.org/

In the long term, one could work on adapting our approach to the Industry of the Future with challenges on interoperability, heterogeneity of protocols, wireless protocols, and adapting our distributed intrusion detection methodology to the Internet of Things (IoT). In this context it could be necessary to investigate the development of a standard document for global system requirements. In our approach, we relied on routing sheets, which are neither a standard document, nor extendable to other industrial sectors. Investigating a high level requirement framework for industrial systems would allow to generalize our approach to any system, from any sector.

Appendixes

A) Programmable Logic Controllers (PLCs)

Programmable Logic Controllers (PLCs) are at the heart of modern ICSs. It is a real-time, modular computing system, with tailored memory and computing resources, adapted to harsh industrial environments. A PLC includes the following components:

- Rack. It is the support between modules. It encompasses a communication bus in order to allow communication between all the modules.
- Power supply module.
- Processing module (Central Processing Unit).
- Memory module (Random-Access Memory (RAM) unit).
- Input module (digital or analog).
- Output module (digital or analog).
- Mixed I/O modules.

The functioning of a PLC consists in a processing cycle that includes four activities: (i) inputs reading (sensors), (ii) calculations (execute control logic), (iii) outputs writing (actuators) and (iv) communication (use of communication protocols). Step (ii) corresponds to calculations of the users' program that can be written in various languages. IEC 61131-3 standard [84] identified five languages for programming PLCs, namely Structured Text (ST), Sequential Function Charts (SFC), Ladder Logic Diagram (LLD), Function Block Diagram (FBD) and Instruction List (IL). The choice in the language depends on the company culture, the application domain and the programmers' habits.

B) Routing sheets for the manufacturing plant use case

Routing Sheet **Color** METAL	Part 

No	**Operation**	**Workstation**
01	Remove	Remover 1

Table 12: Routing Sheet - Metal part

Routing Sheet **Color** GREEN	Part 

No	**Operation**	**Workstation**
01	Cut a circle on part	Cartesian Robot 1
02	Remove	Remover 2

Table 13: Routing Sheet - Green part

Routing Sheet **Color** BLUE	Part 

No	**Operation**	**Workstation**
01	Cut a circle on part	Cartesian Robot 1
02	Move part from cartesian robot 1 to cartesian robot 2	Two axis Pick&Place robot
03	Cut a square on part	Cartesian Robot 2
04	Remove	Remover 2

Table 14: Routing Sheet - Blue part

Glossary

Automatic Identification System Self-reporting system that uses the VHF (Very High Frequency) radio link for unique identification of a ship. 70

Commercial-Off-The-Shelf Equipment ready-to-use upon installation, usually delivered from a third party vendor. 72, 96

Computer Integrated Manufacturing The Purdue Reference Model for CIM, as it is commonly referred to, is designed to discuss the overall generic functional requirements of any manufacturing facility, regardless of industry, that are amenable to computerization within the foreseeable future and to define the viable relationships between these "automatable" functions and the other many functions of a manufacturing system. [definition from [155]]. 8

Hardware-In-the-Loop System that allows to virtualize the physical process. Electronic cards are used to connect a software process simulator to PLCs. 71, 72, 76

Local loop Elementary building block of a complex system (generally composed of sensors, actuators and a local controller). 10, 11

Remote Terminal Unit Any slave equipment in a Master/Slave architecture (PLC, field equipment, HMI, etc.). 11, 72, 74, 127

Runtime verification *(also known as runtime monitoring)* – Formal verification technique based on extracting information from a running system and using it to detect observed behaviours satisfying or violating certain properties. 23, 43–45, 54, 58, 60, 63, 86, 115, 126, 127

TCP/IP Set of communication protocols used in the internet or similar computer networks. TCP stands for Transmission Control Protocol and IP for Internet Protocol. 11, 17, 61

Acronyms

ANSSI Agence Nationale de la sécurité des systèmes d'information. 8

AP Atomic Proposition. 45

API Application programming interface. 33

APT Advanced Persistent Threat. 14

CAN Controller Area Network. 11, 42, 72, 93, 94, 96, 97, 100, 102, 103, 106, 127

CPS Cyber Physical System. 8

CPU Central Processing Unit. 26, 28, 32

CR Cyber Range. 68, 70, 76

DCS Distributed Control System. 8

DFA Deterministic Finite Automaton. 30, 38–41

DMZ Demilitarized Zone. 20

DPA Deterministic Probabilistic Automaton. 31, 39, 41

DT Digital Twin. 68, 76

DTMC Discrete-Time Markov Chain. 30

FSM Finite-State Machine. xv, xvi, 27, 34, 35, 49, 51, 52, 84–87, 91–93, 97, 100, 106, 116, 117

HIDS Host-based Intrusion Detection System. 24, 28, 43

HMI Human Machine Interface. 10, 30, 36, 68, 71, 72, 74, 90, 91, 94–97, 103, 104, 106, III

I/O Input/Output. 69, 71, I

ICS Industrial Control System. 7–15, 17, 18, 20, 21, 23, 24, 26–31, 33–35, 41–43, 45, 48, 51, 52, 54, 55, 57, 59–63, 67–69, 71, 72, 76–79, 88, 89, 100, 104, 107, 117, 118, 121, 125–127, I

IDN Intrusion Detection Node. 110–112, 114–118, 121–123

IDS Intrusion Detection System. xvi, 20, 23–27, 29, 34–36, 42, 43, 55, 59–62, 67, 71, 74, 75, 80, 98, 99, 102, 109–111, 121, 127

IoT Internet of Things. 128

IT Information Technology. 7, 8, 11–14, 17, 18, 20, 25–27, 68, 70, 125

LAN Local Area Network. 59, 113

LTL Linear Temporal Logic. 36, 45–49, 51, 53, 54, 58, 63, 86, 87, 106, 115, 126, 127

MIS Management Information System. 9, 12, 13, 74

ML Machine Learning. 25–27, 31, 34, 37, 41, 127

MTL Metric Temporal Logic. 36, 45–48, 53, 54, 63, 86, 106, 126, 127

NIDS Network-based Intrusion Detection System. xvii, 24, 28, 34, 43, 74, 76, 117

NIST National Institute of Standards and Technology. 7, 8

NTP Network Time Protocol. 56

OPC Open Platform Communications. 33, 39

OS Operating System. 12

OSI Open Systems Interconnection. 75, 76

OT Operational Technology. 7, 8, 11–13, 17, 20, 68, 70, 125

PID Proportional Integral Derivative. 11, 91

PLC Programmable Logic Controller. 9–11, 14, 17, 18, 24, 30, 32–36, 41, 42, 68–72, 74, 90, 91, 93–97, 101, 102, 104, 112, 113, 118, 121, 122, I, III

PST Probabilistic Suffix Tree. 31, 32

PTP Precision Time Protocol. 56

SCADA Supervisory Control And Data Acquisition. 8–11, 27, 33, 37–39, 42, 60, 69–71

SNTP Simple Network Time Protocol. 117, 118

SSH Secure Shell. 117

STL Signal Temporal Logic. 36, 45, 47, 48, 53, 54, 63, 86, 106, 126, 127

TCP Transmission Control Protocol. 74, 107, 121

TL Temporal Logic. 36, 44, 45, 47–49, 51, 52, 58, 62, 63, 87, 98, 106, 126

VM Virtual Machine. 70, 74

Bibliography

[1] Marshall Abrams and Joe Weiss. Malicious Control System Cyber Security Attack Case Study – Maroochy Water Services, Australia. Technical report, MITRE Corporation, 2008. 14

[2] Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfsdóttir, and Karoliina Lehtinen. An Operational Guide to Monitorability. In *International Conference on Software Engineering and Formal Methods*, pages 433–453. Springer, 2019. 50

[3] Sridhar Adepu, Nandha Kumar Kandasamy, and Aditya Mathur. EPIC: An Electric Power Testbed for Research and Training in Cyber Physical Systems Security. In *Computer Security: ESORICS International Workshops, CyberICPS and SECPRE, Barcelona, Spain, September 6–7, 2018, Revised Selected Papers*, pages 37–52. Springer, 2019. 69

[4] Irfan Ahmed, Vassil Roussev, William Johnson, Saranyan Senthivel, and Sneha Sudhakaran. A SCADA System Testbed for Cybersecurity and Forensic Research and Pedagogy. In *Proceedings of the 2nd Annual Industrial Control System Security Workshop*, pages 1–9. ACM, 2016. 69

[5] Youssif Al-Nashif, Aarthi Arun Kumar, Salim Hariri, Yi Luo, Ferenc Szidarovsky, and Guangzhi Qu. Multi-Level Intrusion Detection System (ML-IDS). In *International Conference on Autonomic Computing*, pages 131–140. IEEE, 2008. 31, 37

[6] Faisal Alsakran, Gueltoum Bendiab, Stavros Shiaeles, and Nicholas Kolokotronis. Intrusion Detection Systems for Smart Home IoT Devices: Experimental Comparison Study. In *International Symposium on Security in Computing and Communication*, pages 87–98. Springer, 2019. 75

[7] Prashant Anantharaman, Anmol Chachra, Shikhar Sinha, Michael Millian, Bogdan Copos, Sean Smith, and Michael Locasto. A Communications Validity Detector for SCADA Networks. In *ICCIP, International Conference on Critical Infrastructure Protection*, pages 155–183. Springer, 2022. 30, 37, 60

[8] Anders Andreasen. Evaluation of an Open-source Chemical Process Simulator Using a Plant-wide Oil and Gas Separation Plant Flowsheet Model as Basis. *Periodica Polytechnica Chemical Engineering*, 66(3):503–511, 2022. 69

[9] Giddeon Angafor, Iryna Yevseyeva, and Ying He. Bridging the Cyber Security Skills Gap: Using Tabletop Exercises to Solve the CSSG Crisis. In *Joint International Conference on Serious Games*, pages 117–131. Springer, 2020. 67

[10] Uchenna Daniel Ani, Hongmei He, and Ashutosh Tiwari. Review of cybersecurity issues in industrial critical infrastructure: manufacturing in perspective. *Journal of Cyber Security Technology*, 1(1):32–74, 2017. 13

[11] Tomáš Babiak, Mojmír Křetínskỳ, Vojtěch Řehák, and Jan Strejček. LTL to Büchi automata translation: Fast and more deterministic. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 95–109. Springer, 2012. 98

[12] Anton Baláž, Jana Trelová, and Matej Kostráb. Architecture of Distributed Intrusion Detection System Based on Anomalies. In *IEEE 14th International Conference on Intelligent Engineering Systems*, pages 79–83. IEEE, 2010. 37, 59

[13] Paul Baran. On Distributed Communications Networks. *IEEE transactions on Communications Systems*, 12(1):1–9, 1964. 55

[14] Rafael Ramos Regis Barbosa, Ramin Sadre, and Aiko Pras. Towards Periodicity Based Anomaly Detection in SCADA Networks. In *IEEE 17th International Conference on Emerging Technologies & Factory Automation (ETFA)*, pages 1–4. IEEE, 2012. 31, 37

[15] Rafael Ramos Regis Barbosa, Ramin Sadre, and Aiko Pras. Exploiting Traffic Periodicity in Industrial Control Networks. *International journal of critical infrastructure protection*, 13:52–62, 2016. 31, 37

[16] Ezio Bartocci, Jyotirmoy Deshmukh, Alexandre Donzé, Georgios Fainekos, Oded Maler, Dejan Ničković, and Sriram Sankaranarayanan. Specification-Based Monitoring of Cyber-Physical Systems: A Survey on Theory, Tools and Applications. *Lectures on Runtime Verification: Introductory and Advanced Topics*, 10457:135–175, 2018. 43, 48

[17] Ezio Bartocci, Yliès Falcone, Adrian Francalanza, and Giles Reger. Introduction to Runtime Verification. In *Lectures on Runtime Verification*, pages 1–33. Springer, 2018. 43, 44, 86

[18] David Basin, Felix Klaedtke, and Eugen Zălinescu. Algorithms for Monitoring Real-time Properties. *Acta informatica*, 55:309–338, 2018. 46

[19] Andreas Bauer and Yliès Falcone. Decentralised LTL Monitoring. *Formal Methods in System Design*, 48:46–93, 2016. 58

[20] Andreas Bauer, Martin Leucker, and Christian Schallhart. Monitoring of Real-time Properties. In *Foundations of Software Technology and Theoretical Computer Science: 26th International Conference, Kolkata, India*, pages 260–272. Springer, 2006. 49, 50, 51

[21] Andreas Bauer, Martin Leucker, and Christian Schallhart. Comparing LTL Semantics for Runtime Verification. *Journal of Logic and Computation*, 20(3):651–674, 2010. 49, 50, 52, 88

[22] Andreas Bauer, Martin Leucker, and Christian Schallhart. Runtime Verification for LTL and TLTL. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 20(4):1–64, 2011. 50, 98

[23] Thomas Becmeur, Xavier Boudvin, David Brosset, Gaël Héno, Thibaud Merien, Olivier Jacq, Yvon Kermarrec, and Bastien Sultan. A platform for raising awareness on cyber security in a maritime context. In *International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 103–108. IEEE, 2017. 70

[24] Arupjyoti Bhuyan. Idaho National Laboratory (INL) Wireless Test beds for Over the Air (OTA) experimentation, 2022. [Online] https://inldigitallibrary.inl.gov/sites/sti/sti/Sort_63140.pdf, last accessed Apr. 2024. 69

[25] Andrei Borshchev. *The big book of simulation modeling: multimethod modeling with AnyLogic 6*. AnyLogic, 2013. 69

[26] Andrea Carcano, Alessio Coletta, Michele Guglielmi, Marcelo Masera, Igor Nai Fovino, and Alberto Trombetta. A Multidimensional Critical State Analysis for Detecting Intrusions in SCADA Systems. *IEEE Transactions on Industrial Informatics*, 7(2):179–186, 2011. 34, 37, 127

[27] Marco Caselli, Emmanuele Zambon, and Frank Kargl. Sequence-aware Intrusion Detection in Industrial Control Systems. In *Proceedings of the 1st ACM Workshop on Cyber-Physical System Security*, pages 13–24, 2015. 28, 30, 31, 37

[28] Christos Cassandras and Stephane Lafortune. *Introduction to Discrete Event Systems*. Springer, 2nd edition, 2010. 11

[29] Steven Cheung, Bruno Dutertre, Martin Fong, Ulf Lindqvist, Keith Skinner, and Alfonso Valdes. Using Model-based Intrusion Detection for SCADA Networks. In *Proceedings of the SCADA security scientific symposium*, volume 46, pages 1–12. SRI International, 2007. 29, 30, 37

[30] Seungoh Choi, Jongwon Choi, Jeong-Han Yun, Byung-Gil Min, and HyoungChun Kim. Expansion of ICS Testbed for Security Validation based on MITRE ATT&CK Techniques. In *13th USENIX Workshop on Cyber Security Experimentation and Test (CSET)*, 2020. 69

[31] Justyna Chromik, Anne Remke, and Boudewijn Haverkort. Bro in SCADA: dynamic intrusion detection policies based on a system model. In *5th International Symposium for ICS & SCADA Cyber Security Research*, pages 112–121, 2018. 37

[32] CiA 301-4.2.0. CANopen application layer and communication profile. Public available specification, CAN in Automation, 2011. 93

[33] CiA 404-1.2.0. CANopen profile for measuring devices and closed-loop controllers. Public available specification, CAN in Automation, 2002. 84

[34] Christian Colombo and Yliès Falcone. Organizing LTL Monitors over Distributed Systems wth a Global Clock. *Formal Methods in System Design*, 49:109–158, 2016. 58, 115, 127

[35] Mauro Conti, Denis Donadel, and Federico Turrin. A survey on Industrial Control System Testbeds and Datasets for Security Research. *IEEE Communications Surveys & Tutorials*, 23(4):2248–2294, 2021. 68, 69

[36] Darkweb and Nnovic. ModbusPal - a Java MODBUS simulator. [Online] https://sourceforge.net/projects/modbuspal/, last accessed Apr. 2024. 69

[37] Agence Nationale de la Sécurité des Systèmes d'Information. *Cybersecurity for Industrial Control Systems - Managing Cybersecurity for Industrial Control Systems*. ANSSI, 2012. 12, 17

[38] Agence Nationale de la Sécurité des Systèmes d'Information. *Cybersecurity for Industrial Control Systems - Classification Method and Key Measures*. ANSSI, 2014. 8

[39] Hervé Debar, Marc Dacier, and Andreas Wespi. Towards a taxonomy of intrusion-detection systems. *Computer networks*, 31(8):805–822, 1999. 24

[40] Hervé Debar, Benjamin Morin, Frédéric Cuppens, Fabien Autrel, Ludovic Mé, Bernard Vivinis, Salem Benferhat, Mireille Ducassé, and Rodolphe Ortalo. Détection d'intrusions: corrélation d'alertes. *Revue des Sciences et Technologies de l'Information-Série TSI: Technique et Science Informatiques*, 23:359–390, 2004. 127

[41] Hervé Debar, Marc Dacier, and Andreas Wespi. A revised taxonomy for intrusion-detection systems. *Annales Des Télécommunications*, 55(7-8):361–378, 2000. 24

[42] Dorothy E Denning. An Intrusion-Detection Model. *IEEE Transactions on software engineering*, 13(2):222–232, 1987. 24

[43] Jack Depperschmidt. Idaho National Laboratory (INL) Smart Grid Test Bed Revision 2. Technical report, INL, 2017. [Online] https://www.energy.gov/sites/prod/files/2017/10/f38/CX-270322.pdf, last accessed Apr. 2024. 69

[44] WINGPATH Software Development. ModMultiSim - Programmable Modbus simulator for multiple devices. [Online] https://wingpath.co.uk/modbus/modmultisim.php, last accessed Apr. 2024. 69

[45] Talat Dogan. Chemical Process Simulation for Everyone, 2019. [Online] https://control.com/forums/threads/a-new-modbus-tcp-server-simulator.47713/, last accessed Apr. 2024. 69

[46] Wenli Duo, MengChu Zhou, and Abdullah Abusorrah. A Survey of Cyber Attacks on Cyber Physical Systems: Recent Advances and Challenges. *IEEE/CAA Journal of Automatica Sinica*, 9(5):784–800, 2022. 8

[47] Matthew B Dwyer, George S Avrunin, and James C Corbett. Patterns in property specifications for finite-state verification. In *Proceedings of the 21st international conference on Software engineering*, pages 411–420, 1999. 52, 53, 85

[48] Cindy Eisner, Dana Fisman, John Havlicek, Yoad Lustig, Anthony McIsaac, and David Van Campenhout. Reasoning with Temporal Logic on Truncated Paths. In *Computer Aided Verification: 15th International Conference, CAV, Boulder, CO, USA*, pages 27–39. Springer, 2003. 49, 50

[49] Allen Emerson. Temporal and modal logic. In *Formal Models and Semantics*, pages 995–1072. Elsevier, 1990. 49

[50] Noam Erez and Avishai Wool. Control variable classification, modeling and anomaly detection in Modbus/TCP SCADA systems. *International Journal of Critical Infrastructure Protection*, 10:59–70, 2015. 33, 37

[51] Georgios E Fainekos and George J Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science*, 410(42):4262–4291, 2009. 47, 127

[52] Yliès Falcone, Srđan Krstić, Giles Reger, and Dmitriy Traytel. A taxonomy for classifying runtime verification tools. *International Journal on Software Tools for Technology Transfer*, 23(2):255–284, 2021. 98

[53] Nicolas Falliere, Liam O Murchu, and Eric Chien. W32.stuxnet dossier. Technical report, Symantec Security Response, 2011. [Online] https://docs.broadcom.com/doc/security-response-w32-stuxnet-dossier-11-en, last accessed Apr. 2024. 14

[54] Davide Fauri, Daniel Ricardo Dos Santos, Elisa Costante, Jerry den Hartog, Sandro Etalle, and Stefano Tonetta. From System Specification to Anomaly Detection (and Back). In *Workshop on Cyber-Physical Systems Security and Privacy (CPS)*, pages 13–24. ACM, 2017. 36, 38, 42

[55] Benedikt Ferling, Justyna Chromik, Marco Caselli, and Anne Remke. Intrusion Detection for Sequence-Based Attacks with Reduced Traffic Models. In *Measurement, Modelling and Evaluation of Computing Systems: 19th International GI/ITG Conference, Erlangen, Germany*, pages 53–67. Springer, 2018. 30, 38

[56] Dominik Fisch, Alexander Hofmann, Valentin Hornik, Ivan Dedinski, and Bernhard Sick. A Framework for Large-Scale Simulation of Collaborative Intrusion Detection Systems. In *IEEE Conference on Soft Computing in Industrial Applications*, pages 125–130. IEEE, 2008. 57, 60

[57] Robert Flosbach, Justyna Joanna Chromik, and Anne Remke. Architecture and prototype implementation for process-aware intrusion detection in electrical grids. In *38th Symposium on Reliable Distributed Systems (SRDS)*, pages 42–4209. IEEE, 2019. 35, 38

[58] Joint Task Force. Risk Management Framework for Information Systems and Organizations: A System Life Cycle Approach for Security and Privacy. Technical report, NIST Special Publication 800-37 Rev. 2, 2018. [Online] https://doi.org/10.6028/NIST.SP.800-37r2, last accessed Apr. 2024. 18

[59] Igor Nai Fovino, Andrea Carcano, Thibault De Lacheze Murel, Alberto Trombetta, and Marcelo Masera. Modbus/DNP3 State-Based Intrusion Detection System. In *24th IEEE International Conference on Advanced Information Networking and Applications*, pages 729–736. IEEE, 2010. 34, 38

[60] Igor Nai Fovino, Alessio Coletta, Andrea Carcano, and Marcelo Masera. Critical State-Based Filtering System for Securing SCADA Network Protocols. *IEEE Transactions on industrial electronics*, 59(10):3943–3950, 2012. 34, 38, 42

[61] Haihui Gao, Yong Peng, Kebin Jia, Zhonghua Dai, and Ting Wang. The design of ics testbed based on emulation, physical, and simulation (eps-ics testbed). In *Ninth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, pages 420–423. IEEE, 2013. 69

[62] Wei Gao, Thomas Morris, Bradley Reaves, and Drew Richey. On SCADA Control System Command and Response Injection and Intrusion Detection. In *2010 eCrime Researchers Summit*, pages 1–9. IEEE, 2010. 34, 38

[63] Luis Garcia, Saman Zonouz, Dong Wei, and Leandro Pfleger De Aguiar. Detecting PLC Control Corruption via On-Device Runtime Verification. In *2016 Resilience Week (RWS)*. IEEE, 2016. 32, 38

[64] Pedro Garcia-Teodoro, Jesus Diaz-Verdejo, Gabriel Maciá-Fernández, and Enrique Vázquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *computers & security*, 28(1-2):18–28, 2009. 25, 27, 41

[65] Xiaohua Ge, Fuwen Yang, and Qing-Long Han. Distributed networked control systems: A brief overview. *Information Sciences*, 380:117–131, 2017. 55, 56, 57

[66] Robert Gillen, Laura Ann Anderson, Christopher Craig, Jordan Johnson, Rachel Anderson, Andrew Craig, and Stephen L Scott. Design and implementation of full-scale industrial control system test bed for assessing cyber-security defenses. In *IEEE 21st International Symposium on" A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*, pages 341–346. IEEE, 2020. 69

[67] Jairo Giraldo, David Urbina, Alvaro Cardenas, Junia Valente, Mustafa Faisal, Justin Ruths, Nils Ole Tippenhauer, Henrik Sandberg, and Richard Candell. A Survey of

Physics-Based Attack Detection in Cyber-Physical Systems. *ACM Computing Surveys (CSUR)*, 51(4):1–36, 2018. 28

[68] Niv Goldenberg and Avishai Wool. Accurate modeling of Modbus/TCP for intrusion detection in SCADA systems. *International Journal of Critical Infrastructure Protection*, 6(2):63–75, 2013. 30, 38

[69] Bur Goode. Voice over internet protocol (VoIP). *Proceedings of the IEEE*, 90(9):1495–1517, 2002. 12

[70] Benjamin Green, Anhtuan Lee, Rob Antrobus, Utz Roedig, David Hutchison, and Awais Rashid. Pains, Gains and PLCs: Ten Lessons from Building an Industrial Control Systems Testbed for Security Research. In *10th USENIX workshop on cyber security experimentation and test*, 2017. 69

[71] Marco Grochowski, Stefan Kowalewski, Melanie Buchsbaum, and Christian Brecher. Applying Runtime Monitoring to the Industrial Internet of Things. In *24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 348–355. IEEE, 2019. 36, 38

[72] Vilhelm Gustavsson. Machine Learning for a Network-based Intrusion Detection System, An application using Zeek and the CICIDS2017 dataset, 2019. 75

[73] Dina Hadziosmanovic, Robin Sommer, Emmanuele Zambon, and Pieter Hartel. Through the Eye of the PLC: towards Semantic Security Monitoring for Industrial Control Systems. In *Proc. ACSAC*, volume 14, pages 126–135, 2014. 33, 38

[74] Klaus Havelund and Grigore Roşu. Monitoring Programs Using Rewriting. In *Proceedings 16th Annual International Conference on Automated Software Engineering (ASE)*, pages 135–143. IEEE Computer Society, 2001. 50, 98

[75] Vojtěch Havlena, Petr Matoušek, Ondřej Ryšavỳ, and Lukáš Holík. Accurate Automata-Based Detection of Cyber Threats in Smart Grid Communication. *IEEE Transactions on Smart Grid*, 14, 2023. 31, 39

[76] Raymond C Borges Hink and Katerina Goseva-Popstojanova. Characterization of cyber-attacks aimed at integrated industrial control and enterprise systems: A case study. In *IEEE 17th International Symposium on High Assurance Systems Engineering (HASE)*, pages 149–156. IEEE, 2016. 69

[77] Siegfried Hollerer, Wolfgang Kastner, and Thilo Sauter. Safety and Security: A Field of Tension in Industrial Practice. In *IEEE 21st International Conference on Industrial Informatics (INDIN)*, pages 1–7. IEEE, 2023. 18

[78] David Holmes, Maria Papathanasaki, Leandros Maglaras, Mohamed Amine Ferrag, Surya Nepal, and Helge Janicke. Digital twins and cyber security–solution or challenge? In *6th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM)*, pages 1–8. IEEE, 2021. 68

[79] Estelle Hotellier, Franck Sicard, Julien Francq, and Stéphane Mocanu. Standard Specification-based Intrusion Detection for Hierarchical Industrial Control Systems. *Information Sciences*, 2024. Article 120102. 63

[80] Yan Hu, Hong Li, Hong Yang, Yuyan Sun, Limin Sun, and Zhiliang Wang. Detecting stealthy attacks against industrial control systems based on residual skewness analysis. *EURASIP Journal on Wireless Communications and Networking*, 2019(1):1–14, 2019. 33, 38, 42

[81] IEC 27001. Information security, cybersecurity and privacy protection – Information security management systems. International standard, Industrial Electrotechnical Commission, 2022. 19

[82] IEC 27002. Information security, cybersecurity and privacy protection – Information security controls. International standard, Industrial Electrotechnical Commission, 2022. 19

[83] IEC 27005. Information technology – Security techniques – Information security risk management. International standard, Industrial Electrotechnical Commission, 2022. 18

[84] IEC 61131-3. Programmable controllers - part 3: Programming languages. International standard, International Electrotechnical Commission, 2013. 84, I

[85] IEC 61508. Industrial communication networks - Functional safety of electrical/electronic/programmable electronic safety–related systems. International standard, Industrial Electrotechnical Commission, 2010. 82

[86] IEC 61800. Adjustable speed electrical power drive systems. International standard, Industrial Electrotechnical Commission, 2021. 83, 92

[87] IEC 61850. Communication networks and systems for power utility automation. International standard, Industrial Electrotechnical Commission, 2022. 85, 127

[88] IEC 62264. Enterprise-control system integration. International standard, Industrial Electrotechnical Commission, 2010. 9

[89] IEC 62443. Industrial communication networks - networks and system security. International standard, Industrial Electrotechnical Commission, 2009. 13, 18, 20, 82, 121

[90] Moses Ike, Kandy Phan, Keaton Sadoski, Romuald Valme, and Wenke Lee. SCAPHY: Detecting Modern ICS Attacks by Correlating Behaviors in SCADA and PHYsical. In *IEEE Symposium on Security and Privacy (SP)*, pages 20–37. IEEE, 2023. 33, 39, 42

[91] Kaspersky ICS CERT. Threat landscape for industrial automation systems. Technical report, Kaspersky, 2022. [Online] https://ics-cert.kaspersky.com/media/H1_2019_kaspersky_ICS_REPORT_EN.pdf, last accessed Apr. 2024. 14

[92] Farshad Khorrami, Prashanth Krishnamurthy, and Ramesh Karri. Cybersecurity for Control Systems: A Process-Aware Perspective. *IEEE Design & Test*, 33(5):75–83, 2016. 33, 39

[93] Ansam Khraisat and Ammar Alazab. A critical review of intrusion detection systems in the internet of things: techniques, deployment strategy, validation strategy, attacks, public datasets and challenges. *Cybersecurity*, 4:1–27, 2021. 27

[94] Amit Kleinmann and Avishai Wool. Automatic Construction of Statechart-Based Anomaly Detection Models for Multi-Threaded SCADA via Spectral Analysis. In *Proceedings of the 2Nd ACM Workshop on Cyber-Physical Systems Security and Privacy*, CPS-SPC, pages 1–12. ACM, 2016. 30, 39

[95] Calvin Ko, Manfred Ruschitzka, and Karl Levitt. Execution monitoring of security-critical programs in distributed systems: a specification-based approach. In *Proceedings IEEE Symposium on Security and Privacy*, pages 175–187, 1997. 25

[96] Sascha Konrad and Betty Cheng. Real-time specification patterns. In *Proceedings of the 27th international conference on Software engineering*, pages 372–381. ACM, 2005. 53, 94

[97] Oualid Koucham, Stéphane Mocanu, Guillaume Hiet, Jean-Marc Thiriet, and Frédéric Majorczyk. Detecting Process-Aware Attacks in Sequential Control Systems. In *Secure IT Systems: 21st Nordic Conference, NordSec, Oulu, Finland*, pages 20–36. Springer, 2016. 39

[98] Oualid Koucham, Stéphane Mocanu, Guillaume Hiet, Jean-Marc Thiriet, and Frédéric Majorczyk. Efficient Mining of Temporal Safety Properties for Intrusion Detection in Industrial Control Systems. In *SAFEPROCESS*, pages 1–8, 2018. 36, 39

[99] Georgia Koutsandria, Reinhard Gentz, Mahdi Jamei, Anna Scaglione, Sean Peisert, and Chuck McParland. A Real-Time Testbed Environment for Cyber-Physical Security on the Power Grid. In *Proceedings of the First ACM Workshop on Cyber-Physical Systems-Security and/or PrivaCy*, pages 67–78, 2015. 69

[100] Siwar Kriaa, Ludovic Pietre-Cambacedes, Marc Bouissou, and Yoran Halgand. A survey of approaches combining safety and security for industrial control systems. *Reliability Engineering and System Safety*, 139:156–178, 2015. 18, 82

[101] Jay Lee, Behrad Bagheri, and Hung-An Kao. A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems. *Manufacturing letters*, 3:18–23, 2015. 8

[102] Wenjuan Li, Weizhi Meng, and Lam For Kwok. Surveying Trust-Based Collaborative Intrusion Detection: State-of-the-Art, Challenges and Future Directions. *IEEE Communications Surveys & Tutorials*, 24(1):280–305, 2021. 55

[103] Chih-Yuan Lin and Simin Nadjm-Tehrani. Understanding IEC-60870-5-104 traffic patterns in SCADA networks. In *Proceedings of the 4th ACM Workshop on Cyber-Physical System Security*, pages 51–60, 2018. 31, 39, 42

[104] Chih-Yuan Lin and Simin Nadjm-Tehrani. Protocol study and anomaly detection for server-driven traffic in SCADA networks. *International Journal of Critical Infrastructure Protection*, 2023. Article 100612. 32, 39

[105] Yang Liu, Yu Peng, Bailing Wang, Sirui Yao, and Zihe Liu. Review on Cyber-physical Cystems. *IEEE/CAA Journal of Automatica Sinica*, 4(1):27–40, 2017. 8

[106] Mark Luchs and Christian Doerr. Last Line of Defense: A Novel IDS Approach Against Advanced Threats in Industrial Control Systems. In *Detection of Intrusions and Malware, and Vulnerability Assessment: 14th International Conference, DIMVA, Bonn, Germany*, pages 141–160. Springer, 2017. 26

[107] Oded Maler and Dejan Ničković. Monitoring Temporal Properties of Continuous Signals. In *International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 152–166. Springer, 2004. 53

[108] Oded Maler and Dejan Ničković. Monitoring properties of analog and mixed-signal circuits. *International Journal on Software Tools for Technology Transfer*, 15:247–268, 2013. 47, 53, 98

[109] Aditya Mathur and Nils Ole Tippenhauer. SWaT: A water treatment testbed for research and training on ICS security. In *International Workshop on Cyber-Physical Systems for Smart Water Networks (CySWater)*, pages 31–36. IEEE, 2016. 69

[110] Stephen McLaughlin. CPS: Stateful policy enforcement for control system device usage. In *Proceedings of the 29th Annual Computer Security Applications Conference*, pages 109–118. ACM, 2013. 32, 39

[111] Stephen McLaughlin. Blocking Unsafe Behaviors in Control Systems through Static and Dynamic Policy Enforcement. In *Proceedings of the 52nd Annual Design Automation Conference*, pages 1–6, 2015. 32, 39, 42

[112] Kelei Miao, Xiufang Shi, and Wen-An Zhang. Attack signal estimation for intrusion detection in industrial control system. *Computers & Security*, 96, 2020. Article 101926. 33, 39

[113] Robert Mitchell and Ing-Ray Chen. A Survey of Intrusion Detection Techniques for Cyber-physical Systems. *ACM Computing Surveys (CSUR)*, 46(4):1–29, 2014. 25, 27

[114] Robert Mitchell and Ray Chen. Behavior-Rule Based Intrusion Detection Systems for Safety Critical Smart Grid Applications. *IEEE Transactions on Smart Grid*, 4(3):1254–1263, 2013. 39

[115] Robert Mitchell and Ray Chen. Behavior Rule Specification-Based Intrusion Detection for Safety Critical Medical Cyber Physical Systems. *IEEE Transactions on Dependable and Secure Computing*, 12(1):16–30, 2014. 34, 35, 39

[116] MITRE | ATT&CK® for Industrial Control Systems, 2021. [Online] https://attack.mitre.org/matrices/ics/, last accessed Apr. 2024. 15, 78

[117] Sathya Narayana Mohan, Gelli Ravikumar, and Manimaran Govindarasu. Distributed Intrusion Detection System using Semantic-based Rules for SCADA in Smart Grid. In *IEEE/PES Transmission and Distribution Conference and Exposition (T&D)*, pages 1–5. IEEE, 2020. 59

[118] Menna Mostafa and Borzoo Bonakdarpour. Decentralized Runtime Verification of LTL Specifications in Distributed Systems. In *IEEE International Parallel and Distributed Processing Symposium*, pages 494–503. IEEE, 2015. 58

[119] David Myers, Kenneth Radke, Suriadi Suriadi, and Ernest Foo. Process Discovery for Industrial Control System Cyber Attack Detection. In *ICT Systems Security and Privacy Protection: 32nd IFIP TC 11 International Conference*, pages 61–75. Springer, 2017. 33, 39

[120] Gorby Kabasele Ndonda and Ramin Sadre. Exploiting the Temporal Behavior of State Transitions for Intrusion Detection in ICS/SCADA. *IEEE Access*, 10:111171–111187, 2022. 36, 40

[121] Dejan Ničković and Nir Piterman. From MTL to Deterministic Timed Automata. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 152–167. Springer, 2010. 51

[122] Jeyasingam Nivethan and Mauricio Papa. A SCADA Intrusion Detection Framework that Incorporates Process Semantics. In *Proceedings of the 11th Annual Cyber and Information Security Research Conference*, pages 1–5. ACM, 2016. 39, 42

[123] Livinus Obiora Nweke. A Survey of Specification-based Intrusion Detection Techniques for Cyber-Physical Systems. *International Journal of Advanced Computer Science and Applications (IJACSA)*, 12(5):37–45, 2021. 25, 27

[124] Maria Leonor Pacheco, Max von Hippel, Ben Weintraub, Dan Goldwasser, and Cristina Nita-Rotaru. Automated Attack Synthesis by Extracting Finite State Machines from Protocol Specification Documents. In *IEEE Symposium on Security and Privacy (SP)*, pages 51–68. IEEE, 2022. 35, 85

[125] Ruoming Pang, Vern Paxson, Robin Sommer, and Larry Peterson. binpac: A yacc for Writing Application Protocol Parsers. In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, pages 289–300, 2006. 75

[126] Stephen Papa, William Casper, and Suku Nair. A transfer function based intrusion detection system for SCADA systems. In *IEEE Conference on Technologies for Homeland Security (HST)*, pages 93–98. IEEE, 2012. 33, 40

[127] Fabio Pasqualetti, Florian Dörfler, and Francesco Bullo. Attack Detection and Identification in Cyber-Physical Systems. *IEEE Transactions on Automatic Control*, 58(11):2715–2729, 2013. 33, 40

[128] Alexandre Philippot, Bernard Riera, M Koza, Romain Pichard, Ramla Saddem, François Gellot, David Annebicque, and Fabien Emprin. HOME I/O and FACTORY I/O: 2 Pieces of innovative PO simulation software for automation education. In *27th EAEEIE Annual Conference (EAEEIE)*, pages 1–6. IEEE, 2017. 70

[129] PLCopen. PLCopen - for efficiency in automation. Technical specification, PLCopen, 2011. [Online] http://www.plcopen.org/, last accessed Apr. 2024. 84, 92

[130] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (SFCS)*, pages 46–57. IEEE Computer Society, 1977. 45

[131] Amir Pnueli and Roni Rosner. On the Synthesis of an Asynchronous Reactive Module. In *International Colloquium on Automata, Languages, and Programming*, pages 652–671. Springer, 1989. 102

[132] Georgios Potamos, Adamantini Peratikou, and Stavros Stavrou. Towards a maritime cyber range training environment. In *IEEE International Conference on Cyber Security and Resilience (CSR)*, pages 180–185. IEEE, 2021. 70

[133] Maxime Puys, Pierre-Henri Thevenon, and Stéphane Mocanu. Hardware-in-the-loop labs for scada cybersecurity awareness and training. In *Proceedings of the 16th International Conference on Availability, Reliability and Security*, pages 1–10, 2021. 71

[134] Harri Pyykkö, Jarkko Kuusijärvi, Sami Noponen, Sirra Toivonen, and Ville Hinkka. Building a virtual maritime logistics cybersecurity training platform. In *Data Science and Innovation in Supply Chain Management: How Data Transforms the Value Chain. Proceedings of the Hamburg International Conference of Logistics (HICL), Vol. 29*, pages 223–246. Berlin: epubli GmbH, 2020. 70

[135] Slavica V Boštjančič Rakas, Mirjana D Stojanović, and Jasna D Marković-Petrović. A review of research work on network-based SCADA intrusion detection systems. *IEEE Access*, 8:93083–93108, 2020. 25, 27

[136] Giles Reger and Klaus Havelund. What is a trace? a runtime verification perspective. In *International Symposium on Leveraging Applications of Formal Methods*, pages 339–355. Springer, 2016. 43

[137] Franka Schuster and Andreas Paul. A Distributed Intrusion Detection System for Industrial Automation Networks. In *Proceedings of IEEE 17th International Conference on Emerging Technologies & Factory Automation (ETFA)*, pages 1–4. IEEE, 2012. 60

[138] Gus Serino and Ben Miller. Recommendations Following the Oldsmar Water Treatment Facility Cyber Attack. Technical report, Dragos Inc, 2021. [Online] https://www.dragos.com/blog/industry-news/recommendations-following-the-oldsmar-water-treatment-facility-cyber-attack/, last accessed Apr. 2024. 15

[139] Vishal Sharma, Ilsun You, Kangbin Yim, Ray Chen, and Jin-Hee Cho. BRIoT: Behavior Rule Specification-based Misbehavior Detection for IoT-Embedded Cyber-Physical Systems. *IEEE Access*, 7:118556–118580, 2019. 35, 40

[140] Franck Sicard, Estelle Hotellier, and Julien Francq. An industrial control system physical testbed for naval defense cybersecurity research. In *IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 413–422. IEEE, 2022. 71

[141] Franck Sicard, Éric Zamaï, and Jean-Marie Flaus. Filters based Approach with Temporal and Combinational Constraints for Cybersecurity of Industrial Control Systems. *International Federation of Automatic and Control (IFAC)*, 51(24):96–103, 2018. 40

[142] Franck Sicard, Éric Zamaï, and Jean-Marie Flaus. An approach based on behavioral models and critical states distance notion for improving cybersecurity of industrial control systems. *Reliability Engineering & System Safety*, 188:584–603, 2019. 35, 40, 42, 127

[143] The Open Source Chemical Process Simulator. Chemical Process Simulation for Everyone. [Online] https://dwsim.org/, last accessed Apr. 2024. 69

[144] Joe Slowik. CRASHOVERRIDE: Analysis of the Threat to Electric Grid Operations. Technical report, Dragos Inc, 2019. [Online] https://www.dragos.com/wp-content/uploads/CRASHOVERRIDE.pdf, last accessed Apr. 2024. 16

[145] Steven R Snapp, James Brentano, Gihan V Dias, Terrance L Goan, Todd Heberlein, Che-Lin Ho, Karl Levitt, Biswanath Mukherjee, Stephen Smaha, Tim Grance, Daniel Teal, and Doug Mansur. DIDS (Distributed Intrusion Detection System)–Motivation, Architecture, and An Early Prototype. In *InProceedings of the 14th National Computer Security Conference*, pages 167–176, 1991. 59

[146] Keith Stouffer, Suzanne Lightman, Victoria Pillitteri, Marshall Abrams, and Adam Hahn. Guide to industrial control systems (ics) security. Technical report, NIST Special Publication 800-82 Revision 2, 2015. http://dx.doi.org/10.6028/NIST.SP.800-82r2, last accessed Apr. 2024. 8

[147] Keith Stouffer, Michael Pease, CheeYee Tang, Timothy Zimmerman, Victoria Pillitteri, Suzanne Lightman, Adam Hahn, Stephanie Saravia, Aslam Sherule, and Michael Thompson. Guide to Operational Technology (OT) Security. Technical report, NIST Special Publication 800-82 Rev. 3, 2023. [Online] https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-82r3.pdf, last accessed Apr. 2024. 7, 8, 43

[148] Kimberly Tam, Kevin Forshaw, Kevin Jones, et al. Cyber-SHIP: Developing next generation maritime cyber research capabilities. In *International Conference on Marine Engineering and Technology (ICMET)*. IMarEST, 2019. 70

[149] Kimberly Tam, Kemedi Moara-Nkwe, and Kevin Jones. The Use of Cyber Ranges in the Maritime Context: Assessing maritime-cyber risks, raising awareness, and providing training. *Maritime Technology and Research*, 2020. 70

[150] Christos Tsigkanos, Marcello M Bersani, Pantelis A Frangoudis, and Schahram Dustdar. Edge-Based Runtime Verification for the Internet of Things. *IEEE Transactions on Services Computing*, 15(5):2713–2727, 2021. 36, 40

[151] Dogan Ulus. Online Monitoring of Metric Temporal Logic using Sequential Networks. *arXiv preprint arXiv:1901.00175*, 2019. 106

[152] David I Urbina, Jairo A Giraldo, Alvaro A Cardenas, Nils Ole Tippenhauer, Junia Valente, Mustafa Faisal, Justin Ruths, Richard Candell, and Henrik Sandberg. Limiting the Impact of Stealthy Attacks on Industrial Control Systems. In *Proceedings of ACM SIGSAC conference on computer and communications security*, pages 1092–1105, 2016. 26, 33, 40

[153] Moshe Ya'akov Vardi. An automata-theoretic approach to linear temporal logic. *Banff Higher Order Workshop 1995*, 1996. 51

[154] Emmanouil Vasilomanolakis, Shankar Karuppayah, Max Mühlhäuser, and Mathias Fischer. Taxonomy and Survey of Collaborative Intrusion Detection. *ACM Computing Surveys (CSUR)*, 47(4):1–33, 2015. 55

[155] Theodore J Williams. A Reference Model for Computer Integrated Manufacturing from The Viewpoint of Industrial Automation. *IFAC Proceedings Volumes*, 23(8):281–291, 1990. 8, III

[156] Konrad Wolsing, Lea Thiemt, Christian van Sloun, Eric Wagner, Klaus Wehrle, and Martin Henze. Can Industrial Intrusion Detection Be SIMPLE? In *European Symposium on Research in Computer Security*, pages 574–594. Springer, 2022. 34, 40

[157] Man-Ki Yoon and Gabriela F Ciocarlie. Communication Pattern Monitoring : Improving the Utility of Anomaly Detection for Industrial Control Systems. In *NDSS workshop on security of emerging networking technologies*, 2014. 31, 40

[158] Clinton Young, Habeeb Olufowobi, Gedare Bloom, and Joseph Zambreno. Automotive Intrusion Detection Based on Constant CAN Message Frequencies Across Vehicle Driving Modes. In *Proceedings of the ACM Workshop on Automotive Cybersecurity*, pages 9–14, 2019. 42

[159] Chunjie Zhou, Shuang Huang, Naixue Xiong, Shuang-Hua Yang, Huiyun Li, Yuanqing Qin, and Xuan Li. Design and Analysis of Multimodel-Based Anomaly Intrusion Detection Systems in Industrial Process Automation. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 45(10):1345–1360, 2015. 27

[160] Bonnie Zhu and Shankar Sastry. SCADA-specific intrusion detection/prevention systems: a survey and taxonomy. In *Proceedings of the 1st workshop on secure control systems (SCS)*, volume 11, page 7, 2010. 27

[161] Lenore Zuck. *Past temporal logic*. The Weizmann Institute of Science (Israel), 1986. 49

Scientific Publications

Academic Journal

- Hotellier Estelle, Sicard Franck, Francq Julien, and Mocanu Stéphane. Standard Specification-based Intrusion Detection for Hierarchical Industrial Control Systems. *Information Sciences*, 659:Article 120102. Elsevier, 2024. doi: https://doi.org/10.1016/j.ins.2024.120102.

Workshop with Proceedings

- Sicard Franck, Hotellier Estelle, and Francq Julien. An Industrial Control System Physical Testbed for Naval Defense Cybersecurity Research. In *IEEE European Symposium on Security and Privacy Workshops (Euro S & P W)*, pages 413-422. IEEE, 2022.

Conferences without Proceedings

- Hotellier Estelle. Introduction to a Behavioral Intrusion Detection Approach for Industrial Control Systems. In *Building a Cyber Resilient City, Society, Economy and Infrastructure, Cybersecurity conference*. 2023.

- Hotellier Estelle. Système de détection d'intrusions pour les bus de terrain CAN, avec Zeek. In *BARBHACK Hacking conference*. 2021.

Masterclasses

- Hotellier Estelle. An Innovative Behavioral Approach of Intrusion Detection for Industrial Control Systems. In *FIC, Forum International de la Cybersécurité, Session Masterclass*. 2023.

Other Communications

- Hotellier Estelle, Sicard Franck, Francq Julien, and Mocanu Stéphane. Système de détection des intrusions distribué pour les systèmes industriels. In *RESSI (Rendez-Vous de la Recherche et de l'Enseignement de la Sécurité des Systèmes d'Information)*. 2023.

- Hotellier Estelle, Sicard Franck, Francq Julien, and Mocanu Stéphane. Behavioral Intrusion Detection Methodology for Hybrid Industrial Control Systems. In *RESSI (Rendez-Vous de la Recherche et de l'Enseignement de la Sécurité des Systèmes d'Information)*. 2022.

- Hotellier Estelle. Distributed Intrusion Detection Methodology of Process Oriented Attacks in Industrial Control Systems In *Journée thématique du GT SSLR sur la sécurité des réseaux*. 2021.

Conferences with Proceedings unrelated to the Ph.D. thesis (additional collaborations)

- Verdecchia Roberto, Cruz Luís, Sallou June, Lin Michelle, Wickenden James and Hotellier Estelle. Data-Centric Green AI An Exploratory Empirical Study. In *International Conference on ICT for sustainability (ICT4S)*, pages 35-45. IEEE, 2022.
- Sicard Franck, Hotellier Estelle, Pérez-Olivares Javier, Zamaï Éric. Optimization of Process-Aware Attack Detection for Industrial Control Systems Security. In *25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 889–896. IEEE, 2020.