Intrusion detection for industrial control systems

Oualid Koucham

Communauté
UNIVERSITÉ Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE LA COMMUNAUTÉ UNIVERSITÉ GRENOBLE ALPES

Spécialité : **Automatique-Productique**

Arrêté ministérial : 25 mai 2016

Présentée par

Oualid KOUCHAM

Thèse dirigée par **Jean-Marc THIRIET, Professeur, Université Grenoble Alpes**
et codirigée par **Stéphane MOCANU, Maître de conférences, Institut Polytechnique de Grenoble, Guillaume HIET, Maître de conférences, CentraleSupélec,** et **Frédéric MAJORCZYK, Docteur, DGA**

préparée au sein du **Laboratoire Grenoble Images Parole Signal Automatique (GIPSA-Lab), département Automatique**
dans **l'Ecole Doctorale Electronique, Electrotechnique, Automatique, Traitement du signal (EEATS)**

Détection d'intrusions pour les systèmes de contrôle industriels

Thèse soutenue publiquement le **12 novembre 2018**,
Jury composé de :

Madame Isabelle CHRISMENT
Professeur des Universités, Université de Lorraine, Rapporteur
Monsieur Michel COMBACAU
Professeur des Universités, Université de Toulouse III Paul Sabatier, Rapporteur
Madame Isabel DEMONGODIN
Professeur des Universités, Université d'Aix-Marseille, Présidente, Examinateur
Monsieur Gregory BLANC
Maître de conférences, Institut Mines-Télécom, Examinateur
Monsieur Jean-Marc THIRIET
Professeur, Université Grenoble Alpes, Directeur de thèse
Monsieur Stéphane MOCANU
Maître de conférences, Institut Polytechnique de Grenoble, Encadrant
Monsieur Guillaume HIET
Maître de conférences, CentraleSupélec, Encadrant
Monsieur Frédéric MAJORCZYK
Docteur, DGA, Encadrant

Remerciements

Je tiens à remercier l'ensemble des personnes ayant contribué au bon déroulement de cette thèse.

J'exprime tout d'abord ma reconnaissance à tous les membres du jury ayant aidé à l'évaluation de ces travaux. Je remercie Isabelle Chrisment et Michel Combacau pour leurs précieux retours en tant que rapporteurs, Isabel Demongodin pour avoir accepté de présider le jury, ainsi que Gregory Blanc pour son rôle d'examinateur.

Je tiens aussi à remercier Guillaume Hiet, Frédéric Majorczyk, Stéphane Mocanu et Jean-Marc Thiriet pour leur encadrement scientifique, technique et humain tout au long de ces travaux ainsi que pour leur patiente relecture de ce manuscrit.

Enfin, je remercie les personnes ayant suivi ces travaux côté DGA pour leur confiance.

Résumé L'objectif de ce travail de thèse est le développement de techniques de détection d'intrusions et de corrélation d'alertes spécifiques aux systèmes de contrôle industriels (ICS). Cet intérêt est justifié par l'émergence de menaces informatiques visant les ICS, et la nécessité de détecter des attaques ciblées dont le but est de violer les spécifications sur le comportement correct du processus physique.

Dans la première partie de nos travaux, nous nous sommes intéressés à l'inférence automatique de spécifications pour les systèmes de contrôle séquentiels et ce à des fins de détection d'intrusions. La particularité des systèmes séquentiels réside dans leur logique de contrôle opérant par étapes discrètes. La détection d'intrusions au sein de ces systèmes a été peu étudiée malgré leur omniprésence dans plusieurs domaines d'application.

Dans notre approche, nous avons adopté le formalisme de la logique temporelle linéaire (LTL) et métrique (MTL) permettant de représenter des propriétés temporelles d'ordre qualitatif et quantitatif sur l'état des actionneurs et des capteurs. Un algorithme d'inférence de propriétés a été développé afin d'automatiser la génération des propriétés à partir de motifs de spécifications couvrant les contraintes les plus communes. Cette approche vise à pallier le nombre conséquent de propriétés redondantes inférées par les approches actuelles. Afin d'y remédier, nous ajoutons une étape de pré-sélection permettant de restreindre la recherche des propriétés valides sur des portions non redondantes des traces d'exécution. Nous évaluons notre approche sur un processus physique complexe contrôlé par plusieurs automates programmables industriels et soumis à des attaques orientées processus. Les résultats montrent qu'une réduction importante des propriétés inférées est atteignable tout en maintenant un taux de détection élevé.

Dans la deuxième partie de nos travaux, nous cherchons à combiner l'approche de détection d'intrusions développée dans le premier axe avec des approches de détection d'intrusions classiques. Pour ce faire, nous développons une approche de corrélation tenant compte des spécificités des systèmes industriels en deux points: (i) l'enrichissement et le prétraitement d'alertes venant de domaines différents (cyber et physique), et (ii) la mise au point d'une politique de sélection d'alertes tenant compte du contexte d'exécution du processus physique.

Le premier point part du constat que, dans un système industriel, les alertes qui sont remontées au corrélateur sont caractérisées par des attributs hétérogènes (attributs propres aux domaines cyber et physique). Au sein des approches de corrélation classiques, une étape de prétraitement est généralement utilisée afin de normaliser les alertes et de compléter les champs manquants. Cependant, le cas des systèmes industriels s'avère être plus complexe de par l'hétérogénéité des attributs. Afin d'y remédier, nous utilisons des informations relatives aux protocoles supportés par les contrôleurs et à la distribution des variables du processus au sein des contrôleurs afin d'enrichir les alertes du domaine physique par des informations du domaine cyber. Le deuxième point concerne le développement d'une politique de sélection d'alertes qui adapte dynamiquement les fenêtres de sélection des alertes selon l'évolution des sous-processus. Les résultats de l'évaluation comparée de notre approche par rapport à des approches classiques purement temporelles montrent une meilleure performance sur la base de métriques telles que le taux de réduction des alertes et la complétude des corrélations.

Mots-clés: Détection d'intrusions, systèmes de contrôle industriels, cybersécurité, corrélation d'alertes, vérification à l'exécution, inférence de spécifications

Abstract The objective of this thesis is to develop intrusion detection and alert correlation techniques geared towards industrial control systems (ICS). Our interest is driven by the recent surge in cybersecurity incidents targeting ICS, and the necessity to detect targeted attacks which induce incorrect behavior at the level of the physical process.

In the first part of this work, we develop an approach to automatically infer specifications over the sequential behavior of ICS. In particular, we rely on specification language formalisms such as linear temporal logic (LTL) and metric temporal logic (MTL) to express temporal properties over the state of the actuators and sensors. We develop an algorithm to automatically infer specifications from a set of specification patterns covering the most recurring properties. In particular, our approach aims at reducing the number of redundant and unfalsifiable properties generated by the existing approaches. To do so, we add a pre-selection stage which allows restricting the search for valid properties over non-redundant portions of the execution traces. We evaluate our approach on a complex physical process steered by several controllers under process oriented attacks. Our results show that a significant reduction in the number of inferred properties is possible while achieving high detection rates.

In the second part of this work, we attempt to combine the physical domain intrusion detection approach developed in the first part with more classical cyber domain intrusion detection approaches. In particular, we develop an alert correlation approach which takes into account some specificities of ICS. First, we explore an alert enrichment approach that allows mapping physical domain alerts into the cyber domain. This is motivated by the observation that alerts coming from different domains are characterized by heterogeneous attributes which make any direct comparison of the alerts difficult. Instead, we enrich physical domain alerts with cyber domain attributes given knowledge about the protocols supported by the controllers and the memory mapping of process variables within the controllers.

In this work, we also explore ICS-specific alert selection policies. An alert selection policy defines which alerts will be selected for comparison by the correlator. Classical approaches often rely on sliding, fixed size, temporal windows as a basis for their selection policy. Instead, we argue that given the complex interdependencies between physical subprocesses, agreeing on an alert window size is challenging. Instead, we adopt selection policies that adapt to the state of the physical process by dynamically adjusting the size of the alert windows given the state of the subprocesses within the physical process. Our evaluation results show that our correlator achieves better correlation metrics in comparison with classical temporal based approaches.

Keywords: Intrusion detection, industrial control systems, cybersecurity, alert correlation, runtime verification, specification mining

GIPSA-Lab, 11 Rue des Mathématiques
38400, Saint-Martin-D'Hères, France

Contents

List of Figures

List of Tables

Introduction

This thesis focuses on the cybersecurity of industrial control systems (ICS). An ICS is a set of devices (electrical, mechanical, hydraulic,...) whose interaction controls the behavior of a physical process in order to achieve an industrial objective (manufacturing, transportation of matter and energy, etc.). ICSs serve as the backbone of several critical infrastructures that provide facilities for the generation and distribution of electricity, water treatment and supply, railway transportation networks, and manufacturing applications. Due to their criticality, failures in these systems, whether of accidental or intentional origins, can lead to significant human and economic loss.

The recent history of ICSs is that of an ever-growing convergence with classical information technology (IT) systems. In an effort to drive down costs and provide stakeholders, engineers and operators with seamless access to the industrial plants, ICSs are nowadays interconnected with the Internet and have adopted off-the-shelf IT technology such as TCP/IP networking, standard computer architectures, and common operating systems. Concurrently, many of the security vulnerabilities which marred IT systems have been exported to ICSs. Thus, if such systems were once considered secure because of their isolation, their use of proprietary protocols and peculiar architectures, this is no longer the case as witnessed by the growing number of increasingly sophisticated cybersecurity incidents in the last decades [39, 138, 32]. As a result, nation-states, organizations, and industries have become sensitive to the security threats of ICSs, and have been adamant in their call for the development of adequate security measures to protect ICSs from security breaches.

One of the key aspects which distinguish an ICS from classical IT systems is the presence of a physical process. This has lead to the apparition of a new class of novel threats which target the physical process. The main goal of these threats is to induce incorrect behavior at the level of the physical process. Examples of incorrect behavior might include driving temperatures beyond thresholds, overflowing tanks, or damaging valves through excessive manipulation. We use the term *process-oriented attacks* to refer to the attacks that are related to such threats. Above all, it should be stressed that process-oriented attacks are not merely hypothetical scenarios that are beyond the means of attackers. Many of the recent high-profile security incidents such as the Stuxnet worm [39] and the CrashOverride [138, 32] attacks are instances of such attacks. In this thesis, we focus on the security of ICSs with respect to process-oriented attacks.

Yet, despite their real possibility, it is far from clear whether we currently dispose

of adequate means to secure ICSs against process-oriented attacks. Generally, we can distinguish between *preventive* security measures and *reactive* security measures. Preventive security measures try to preclude the occurrence of a security incident. However, given the complexity of the systems, the ever-increasing number of vulnerabilities, and the limited financial resources, preventing all possible incidents is a daunting task. As a result, reactive security controls are used as a posteriori measures to detect, contain and react to intrusions. In all cases, security controls in ICSs must not disturb the normal functioning of the system by introducing time latencies which might interfere with the control loops, and must also be compatible with the limited computing and memory resources within ICS components. In this thesis, we identify *intrusion detection* as a promising class of reactive security measures whose aim is to automatically detect security incidents by passively monitoring the ICS.

In the last few years, many ICS-oriented intrusion detection approaches have been explored in the research community. However, the following remarks can be made concerning the state of the art on intrusion detection approaches in ICSs. Approaches that attempt to detect process-oriented attacks by relying solely on observations at the level of the cyber domain lead to poor detection results because of their deficient knowledge of the state of the physical process. On the other hand, approaches which monitor the physical process often resort to limited models which are either insufficient to capture the dynamics of the physical process, or demand significant effort and expert knowledge from security operators to build the detection models. With respect to the state of the art in intrusion detection, this thesis defends the following two positions:

I) Detecting process-oriented attacks that generate manifestations both in the cyber and physical domains of the ICS is best performed by using two separate but complementary types of intrusion detection approaches: (i) process-oriented intrusion detection approaches that operate over the state of sensors and actuators in order to detect the physical domain manifestations of the attacks, and (ii) cyber domain oriented intrusion detection approaches that monitor the cyber domain in order to identify the source of the attacks and help operators distinguish between legitimate operations and attacks.

For the type of approaches in (i), the models should be expressive enough to cover the dynamics of the physical process. In particular, one must keep in mind the hybrid nature of ICS which induces both *continuous* and *sequential* behaviors. The continuous behavior is traditionally modeled through differential equations producing continuous trajectories, while the sequential behavior is modeled through discrete event systems producing discrete sequences of logical actuator/sensor states and events. Moreover, despite their expressivity, building the detection models should be automatically performed in order to lighten the burden on the security operators. Finally, care should be taken so that the alerts raised by the models are pertinent and precise so that the operators can react timely.

The type of approaches in (ii) is closely related to traditional intrusion detection solution in IT systems which have been extensively explored in the computer security community.

II) Based on the separation performed above, it is necessary to link manifestations both from the process-oriented intrusion detection approaches and the classical cyber domain approaches in order to fully understand the impact of the attack on the physical process and its source in the cyber domain. This task is closely related to *alert correlation*, a collection of tasks which, among other things, group low-level alerts to afford operators a higher-level view of the events within the system. So far, there have been few attempts at developing adequate alert correlation approaches for ICSs. In particular, classical alert correlation is geared towards alerts coming solely from the IT domains, and thus characterized by a high degree of homogeneity in terms of attributes. On the other hand, the case is more involved in ICSs where alerts can come from different domains with heterogeneous attributes.

Contributions Based on the previous discussion, our main contributions are as follows:

- We propose an intrusion detection approach which relies on automatically mined process specifications to detect process-oriented attacks on the sequential behavior of an ICS. The specifications are synthesized as monitors that read the execution traces and report violations to the operator. A central aspect of our method consists in reducing the number of mined specifications suffering from redundancies. This method provides significant improvements in comparison with naive specification mining approaches.

- We propose an alert correlation approach to link alerts from different intrusion detection systems spanning both the cyber and the physical domains of the ICS. We tackle two limits of traditional alert correlation systems: the pre-processing and enrichment of alerts across the cyber-physical domains and the definition of an ICS-specific alert selection policy which adjusts the alert window to the runtime context of the physical process.

- We evaluate both our intrusion detection and correlation approach using an experimental testbed in a hardware-in-the-loop setting under both process-oriented attacks and legitimate operator manipulations.

Thesis organization This thesis is organized into two main parts, each part is subdivided into two chapters.

The first part provides the background and related work necessary to understand and position our contributions. In Chapter 1, we introduce the necessary background information on ICSs along with their specificities with respect to IT systems. After a general discussion of security issues within ICSs, we explore some of the major security incidents that involve process-oriented attacks. In Chapter 2, we first present a taxonomy of intrusion detection approaches in ICSs. Based on this taxonomy, we provide a critical discussion of the existing approaches. We also introduce work on runtime verification as a pertinent verification approach for our intrusion detection task. Finally, we discuss existing alert correlation approaches and their limitations with respect to ICSs.

The second part introduces our contributions to intrusion detection and alert correlation for ICSs. In Chapter 3, we discuss our intrusion detection approach geared towards process-oriented attacks. We first introduce our threat model, then present our approach before discussing our evaluation methodology and results. In Chapter 4, we present a correlation approach to link alerts from both the process-oriented intrusion detection system developed in Chapter 3, and from more classical cyber-domain intrusion detection systems. In particular, we present our model to map alerts from the physical domain into the cyber domain given knowledge of the controllers and their protocols. We then present our evaluation of the approach and discuss the correlation results.

Part I

Background and Related Work

Chapter 1

Background

Contents

In this chapter, we introduce the necessary background concepts which will be used throughout this work. First, we introduce and contrast industrial control systems (ICS) to traditional information technology (IT) systems. Next, we motivate the need for intrusion detection as a security control and provide a general introduction to intrusion detection. Finally, we present and analyze some relevant attacks on industrial control systems.

1.1 Industrial control systems

In this section, we introduce ICSs (Section 1.1.1), describe their general architecture (Section 1.1.2) and highlight their differences with traditional IT systems (Section 1.1.3).

1.1.1 Terminology

Several organizations have proposed a definition of an industrial control system. The ANSSI, the National Cybersecurity Agency of France, defines an ICS as "*the set of human and material means which aim at controlling or commanding technical plants composed of a set of sensors and actuators*" ([28]). The National Institute of Standards and Technology (NIST) adopts a similar definition by characterizing an ICS as "*the combination of control components (electrical, mechanical, hydraulic, pneumatic, etc.) that act together to achieve an industrial objective (manufacturing, transportation of matter and energy, etc.)*" ([133]). The aim of an ICS is thus the command and control of an industrial process whose domain of application can vary ostensibly. In practice, ICSs can be characterized by [115]: (i) the nature of their operations such as the transformation of raw material into a final product, or the realization of maintenance activities, (ii) the complexity of the operations which might necessitate the combination of several distributed control activities, and (iii) the geographical distribution of the ICS.

As a consequence of this diversity, the term ICS constitutes, in fact, a broad denomination which encompasses several variants. The literature abounds in alternative denominations such as **SCADA** (Supervisory Control And Data Acquisition), **DCS** (Distributed Control System), **IACS** (Industrial Automation and Control Systems), or **PCS** (Process Control System). However, the limits and differences between these denominations are not always explicitly drawn, and establishing clear distinctions can be tricky no less due to the increasing convergence in technologies employed by these variants.

For instance, the ANSSI uses the term SCADA in reference to systems that "*allow for the acquisition and processing of a large quantity of data (remote measures and alarms, etc.) and control industrial equipment (controllers, actuators, sensors, etc.) by issuing remote commands*" ([28]). NIST draws a sharper distinction by referring to "*systems used to control dispersed assets where centralized data acquisition is as important as control*" ([133]). Thus, in the case of SCADA, the accent is on the geographical distribution of the ICS, and the contrast is particularly apparent with respect to DCS which refers to control production systems within the same geographical zone [43, 145].

In the remainder of this chapter, we abstract away from these terminological ambiguities and introduce a general description of architectures and components that are common to the aforementioned ICS variants.

1.1.2 Architecture

ICSs are hierarchical systems whose structure exhibits several levels as depicted in Figure 1.1. In this section, we provide, for each level, a cursory description of its objective along with a discussion of the main components involved in its realization.

Process level (Level 0). This level constitutes the first line of the system in direct interaction with the physical process. Here, instrumentation components such as actuators and sensors are deployed. Sensors convert physical observables (pressure, temperature, etc.) into electrical signals. We distinguish between two types of sensors : (i) discrete

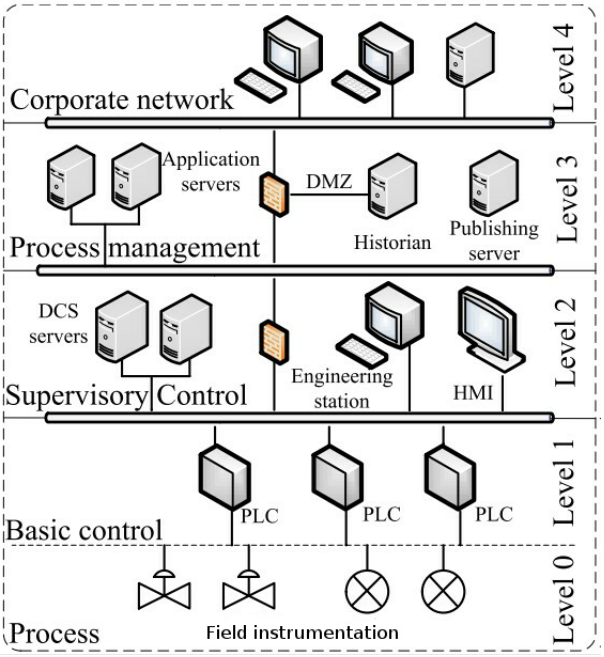


Figure 1.1: The layered architecture of a typical industrial control system [74]

sensors operating on a finite set of values (ex. proximity sensor reporting either the presence or absence of an object), and (ii) analog sensors which allow for continuous measures (ex. pressure sensor returning an output signal proportional to the amount of applied pressure). Actuators typically convert electrical signals into mechanical movements. Similarly to sensors, we can distinguish between discrete (ex. two-way valves) and continuous (ex. DC motors) actuators. Recently, under the name of intelligent electronic devices (IEDs), sensors and actuators have moved towards more elaborate functionalities including the execution of local control operations, the ability to perform diagnostic operations and to communicate with other components [133].

Actuators and sensors exchange data with components at higher levels, typically controllers, either through direct connections through a star architecture, or using a bus architecture. In terms of communication, the volume of exchanged data at this level is low, and exchanges are cyclical with short transfer time. Examples of field-level communication protocols include AS-i bus[1], Profibus DP[2], and Ethercat[3].

Basic control level (Level 1). The objective of this level is to steer and control the physical process in interaction with the field level's instrumentation components. Control is performed using embedded devices known as controllers. The main goal of a controller is the execution of control logics and algorithms necessary for the control of an industrial

[1] http://www.as-interface.net/
[2] https://www.profibus.com/
[3] https://www.ethercat.org/

process. Digital controllers, also known as Programmable Logic Controllers (PLCs), are at the heart of modern industrial control systems as a replacement to traditional electromechanical relays.

Within process control, we can distinguish between continuous and sequential control. Continuous (or regulatory) control aims at maintaining a physical value at the desired level (also called setpoint) by manipulating and correcting control signals sent to the process. A proportional-integral-derivative controller (PID controller) is an example of a widely used closed-loop continuous control mechanism. Sequential control is concerned with issuing ordered sequences of commands to switch the process between different contexts such as startup, shutdown, batching, etc. In practice, both continuous and sequential controls are usually intertwined to achieve the overall control objectives.

In terms of architecture, PLCs include a processor (CPU), random-access memory (RAM), input/output modules through which sensors and actuators can be reached, as well as communication interfaces [36]. While many of these components are common to traditional computing systems, PLCs are characterized by their real-time execution requirements, their tailored memory, and computing resources, as well as their resilience requirements to harsh conditions (high temperatures, electromagnetic interferences, mechanical vibrations, etc.) [36].

The activity of a PLC reduces essentially to the execution of a cycle which includes: (i) reading inputs from sensors, (ii) executing control logics, (iii) transitioning to new states, and (iv) writing outputs to actuators. This cycle can be suspended at any time to handle raised interrupts. The controller then executes the routine associated with the interrupt before resuming the execution cycle. For instance, timer interrupts can be used to guarantee precise sampling times which are necessary for applications such as PID control.

A controller's memory contains two types of data: (i) the program or control logics, and (ii) the state of a set of variables corresponding to setpoints or to the physical process's actuator and sensor states. In terms of control logics programming, the IEC 61131-3 standard [64] defines several programming languages: (i) Ladder language, (ii) Sequential Function Chart (SFC), (iii) Function Block Diagrams (FDB), (iv) Structured Text (ST), and (v) Instruction List (IL). The choice of a language depends on the particular application domain and on geographical or industry-specific differences in language adoption.

With regard to communication, this level includes data exchanges among controllers for distributed control applications, as well as exchanges with the upper supervisory level. Traffic is characterized, albeit in a lesser extent compared to the field level, by its periodicity, its low data volume and its short transfer times. Examples of communication protocols include Modbus[4], DNP3[5], and IEC61850[6].

[4] www.modbus.org/
[5] https://www.dnp.org/
[6] https://webstore.iec.ch/searchform&q=61850

Supervisory control level (Level 2). Operating at a higher hierarchical degree, this level presents a global view of the system's control state and operations. Control servers are responsible for the gathering of information from lower layers for monitoring and diagnostic purposes. This information can be presented to a process operator through a human-machine interface (HMI). An HMI displays graphical indicators that inform the operator about the state of the physical process or the occurrence of alarm notifications. The operator can act on the physical process through the HMI as well. The supervisory level also includes engineering workstations which allow the specification of setpoints and the programming of the controllers. Communication at this level begins to resemble that of traditional IT systems: the volume of data exchanges becomes significant with lower requirements in terms of transfer time. Typical network protocols encountered at this level include OPC DA[7] and OPC UA[8].

Process management and corporate network levels (Level 3-4). The core activities at these levels pertain to the allocation and optimization of resources, maintenance planning, and quality control. These activities are based essentially on information gathered from lower levels. The automatic collection and storage of this information are typically performed by historians which act as a centralized database that might hold configuration parameters as well as process variable states and setpoints.

In the next section, we compare classical IT systems to ICSs to motivate our need to develop ICS-specific intrusion detection and alert correlation approaches.

1.1.3 Comparison between classical IT systems and ICSs

In order to quickly align industrial plants' orientations with economical stakes, plant operators require seamless access to recent and complete information about the plant's state. As a result, the recent history of ICSs is that of a growing convergence with classical information technology (IT). If ICSs were once considered protected from digital threats because of their isolation and reliance on proprietary protocols, as well as their uncommon architectures, such a vision is no longer valid. In practice, ICSs have begun moving towards standard protocols and architectures, commercial operating systems and the use of public networks [115]. However, major differences remain which should not be neglected when thinking about security within ICSs. A simple transposition of IT security measures might not be effective due to certain fundamental divergences :

- **Performance.** ICSs have strong constraints in terms of real-time and deterministic performance [43, 133]. Any delay in data processing or transmission can deteriorate the performance of the control loops. It is also necessary to limit the response time variance of signals (i.e, the jitter) as such behaviors can affect the integral and derivative components of regulators [43]. These strict requirements tend to fade

[7] https://opcfoundation.org/developer-tools/specifications-classic/data-access/
[8] https://opcfoundation.org/developer-tools/specifications-unified-architecture/

as we move towards higher levels of the ICS. In traditional IT systems, response times are less critical and jitter constraints concern only certain applications such as voice over IP. IT systems have also greater requirements in terms of bandwidth compared to industrial networks.

- **Availability.** The nature of industrial systems, in contact with the physical process, requires a high degree of availability. Unpredictable shortages are thus unacceptable. The shutdown of an ICS needs to be planned in advance in order to avoid any negative impact on the production. In comparison, availability in IT systems is less critical.

- **Resource constraints.** Most components in an ICS are limited in terms of resources. In particular, embedded and real-time devices such as controllers are based on tailored hardware. Processing power, memory resources, and network bandwidth are thus not as abundant as in classical IT systems. When combined with the strict requirements of ICSs in terms of real-time and deterministic performance, such constraints hamper the implementation of heavy security controls such as encryption.

- **The lifetime of components.** Spurred by rapid technological advances, IT system components typically have a lifetime in the order of 3 to 5 years. In contrast, ICSs tend to resist changes, leading to much longer lifetimes which can vary between 15 to 20 years or more [133]. Many factors contribute to this disparity including the difficulty to interrupt the functioning of an industrial system without affecting its production, and the heavy investment in terms of time and resources required to test the compatibility to new solutions before their deployment.

From the above discussion, we see that a simple transposition of IT security measures to ICSs is not adequate due to their different characteristics. For instance, while components in IT systems possess in general enough resources to perform cryptographic operations, this is not the case in ICSs where devices have tailored resources, and where heavy computations can disturb the time constraints of the control loops. As a result, we aim in this thesis to explore security measures that do not impact the normal functioning of the ICS. In the next section, we discuss such a security measure.

1.2 Security in ICS

This section explores intrusion detection as a promising reactive security control for ICSs. We first discuss security properties and controls relevant to industrial control systems in Section 1.2.1 with an emphasis on intrusion detection as an interesting security control. Then, Section 1.2.2 provides a general introduction to intrusion detection and to the defining features of intrusion detection systems.

1.2.1 Security properties and controls

Information system security seeks to thwart attacks on data and services by guaranteeing a set of fundamental security properties. While such properties are also relevant in the case of ICSs, their definition and scope can be extended to match the specificities of industrial plants.

- **Confidentiality** which refers to the *non-divulgation of information to unauthorized people or systems*. In ICSs, this amounts to protecting: (i) data relative to performance, planning and set up operations, and (ii) passwords and encryption keys. Moreover, we can also seek to protect data exchanges between instrumentation components at the field level, and PLCs at the control level [21]
- **Integrity** which refers to the *prevention of any modification or destruction of information by unauthorized people or systems*. In ICSs, we can protect the integrity of data sent by sensors, or the integrity of commands. It is also possible to define a notion of **measures integrity** which seeks to protect a sensor against any action that might affect its capacity to send data representative of its deployment environment [21]. For instance, we might physically protect the sensor against displacement or the addition of an element which might perturb its measures.
- **Availability** which *ensures that authorized people or systems can at any time access the services or resources offered by the system*. For ICSs, availability is particularly important especially at the lowest levels of the system (field and control zones).

In terms of desired security properties, we notice an inversion between IT systems and ICSs. Traditionally, the desired security properties for IT systems are, by decreasing order of importance, confidentiality, integrity, and availability. In ICSs, the order is inverted by privileging availability, followed by integrity and confidentiality [21].

The desired security properties are ensured through a *security policy*. The security policy explicitly specifies rules that distinguish authorized from unauthorized behavior. A breach in the security policy is called an *intrusion*. The implementation of the security policy consists of selecting the set of *security controls* able to enforce the security policy. Some security controls try to prevent the occurrence of a security incident. For instance, authentication, access control, system hardening, and secure programming are examples of *preventive* security control measures. However, given the complexity of the systems, the ever-increasing number of vulnerabilities, and the limited financial resources, preventing all possible intrusions is a difficult task. To detect, contain and react to intrusions, *reactive* security controls such as antivirus software have been developed. When implementing a security control in ICSs, one must be careful not to disturb the normal functioning of the system. For instance, security controls must not introduce time latencies which might disturb the control loops, and must also be compatible with the limited computing and memory resources within ICS components. This reactive family of security controls is particularly interesting as it can be deployed on dedicated components and can operate in a totally passive manner [30] to avoid disturbing the ICS. Moreover,

certain classes of intrusion detection approaches can build base profiles of the system's normal behavior in order to detect novel attacks [30]. Such approaches are promising in ICSs due to the relatively simpler network dynamics in terms of fixed topologies, limited user population, and regular communication patterns [21]. In this thesis, we develop an intrusion detection approach geared towards ICSs.

1.2.2 Intrusion detection

Intrusion detection is a reactive or a posteriori security control which seeks to automatically identify violations of the security policy of a monitored system. To perform its task, the intrusion detection system (IDS) can use different *data sources* from the monitored environment. Then, through a *detection method*, the IDS detects the presence of an intrusion and raises an alert. IDS can be classified depending on the type of data source and the detection method [30].

Data source. In terms of data source, we distinguish between *network-based IDS (NIDS)* and *host-based IDS (HIDS)*. NIDS monitor, capture and analyze network data (packets headers, payloads, etc.). Packet analysis can vary in complexity; advanced IDS can perform deep inspection of the packets including the application layers. One advantage of NIDS is that they do not interfere with the performance of hosts when deployed in separate machines. However, network encryption, along with the increase in network bandwidth, can impact the capacities of IDS to efficiently analyze packets.

On the other hand, HIDS rely on host data sources such as operating systems or application logs. Because they rely on host data, HIDS are not affected by the use of network encryption. Application-specific IDS have the advantage of dealing with less varied data (formats specific to the application logs, restricted attacks, etc.) at the cost of a lack in generality. When monitoring OS data, HIDS can use logs from audit tools, system call traces, along with information about users or file systems. In general, HIDS can impact hosts in terms of memory and computing resources. Moreover, a HIDS located at a particular host has a restricted scope and affords limited visibility of intrusions.

Detection method. We can identify two major detection methods: *anomaly-based* and *misuse-based*. Anomaly-based approaches suppose that an intrusion can be detected by observing deviations from the normal behavior of the monitored system. Depending on how knowledge about the system's normal behavior is acquired, we can distinguish between several anomaly-based approaches: (i) approaches based on statistical models or machine learning algorithms, and (ii) approaches based on explicit and often formal specifications of the normal behavior of the system.

The main advantage of anomaly-based approaches is their theoretical capacity to detect unknown attacks. However, these approaches tend to generate a consequent amount of false alerts [45]. This is due to the difficulty of representing exhaustively the normal behavior of the system using limited learning data, or the difficulty to totally specify the behavior of complex systems. Moreover, a system's normal behavior tends to change

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

Figure 1.2: Possibilities for an IDS verdict

with time, which necessitates updates to the base profile. Care should also be taken to use attack-free data when building the base profile. Finally, another weak aspect of such an approach is its poor alert characterization; it is difficult for the operator to understand exactly the nature of the deviation without further context.

Misuse-based approaches rely on a knowledge base of abnormal behavior, usually represented by attack signatures. In general, misuse-base IDS use pattern matching algorithms to recognize suspicious actions. Compared to anomaly-based approaches, this method theoretically generates less false alerts. This, however, assumes that the signatures are pertinent. Also, characterizing an alert is easier since detection relies on a knowledge base of attacks. However, misuse-based approaches are at risk of missing novel attacks. Thus, significant efforts need to be expanded in order to keep signature databases up to date with pertinent signatures. The task of writing attack signatures requires some level of expertise. When writing attack signatures rules, a balance needs to be achieved between too specific rules which might be easily circumvented by an attacker, and overly generic rules that might induce false alerts.

Performance evaluation. The performance of an IDS can be assessed based on the following cases 1.2: (i) true positives (TP) which correspond to correctly identified attacks, (ii) false positives (FP) which correspond to genuine behavior identified as malicious, (iii) true negatives (TN) which correspond to the correct rejection of genuine behavior, and (iv) false negatives (FN) which correspond to missed attacks. Together, these cases allow us to define two metrics to evaluate an IDS performance : (i) True Positive Rate (TPR) which measures the sensitivity of the IDS and is given by $\frac{TP}{(TP+FN)}$ ($= 1$ if no false negative), and (ii) False Positive Rate (FPR) which measures the specificity of the IDS and is given by $\frac{FP}{(FP+TN)}$ ($= 0$ if no false positives).

1.3 Some relevant attacks on industrial control systems

In this section, we discuss some recent attacks on ICSs which are of particular interest for this work. While numerous ICSs specific security incidents have been reported in the last decades[9,10], we focus on attacks where the goal of the attacker is the disruption of the physical process as they are the most relevant to the approach we explore in this thesis.

[9] http://www.risidata.com/Database/

[10] https://ics-cert.us-cert.gov/alerts

Thus, we do not discuss attacks such as Dragonfly/Havex [37] or Blackenergy [65] were espionage and information gathering is the main objective. However, these attacks are not without interest as they can serve as a stepping stone for the more targeted attacks we discuss.

1.3.1 Maroochy Shire

The Maroochy Shire cyber event is the first widely known example of an attacker maliciously breaking into a control system [97]. The attack perpetrator was Vitek Boden, a disgruntled former employee of Hunter Watertech, an Australian company that installed radio-controlled equipment for the Maroochy Shire Council in Queensland, Australia. After failing to land a job with the Maroochy Shire Council, Boden decided to get even with his former employer and the Council. Through a series of attacks between February 29 and April 23, 2000, Boden issued radio commands to the sewage equipment which he might have helped install. Using proper radio equipment attached to a possibly stolen computer, Boden was able to drive around the area and launch his remote attacks which ultimately lead to the spillage of 800,000 liters of raw sewage into local parks and rivers, and the death of marine life. Boden was sentenced to two years in jail and to the reimbursement of the cleanup costs to the Council.

The Maroochy Water Services Sewerage SCADA system consisted of 142 sewage pumping stations, each station containing a computing device able to receive instructions from a central control server and to transmit alarm signals. Communication among pumping stations and between each pumping station and the central server was performed using a two-way radio system. Starting from February 2000, the sewerage system exhibited a series of incorrect behavior including (i) pumps not running as expected, (ii) alarms not being reported to the central server, and (iii) communication loss between the central server and the pumping stations. Technical experts summoned to look into the issue came to the conclusion that the problems experienced by the system were the result of human intervention rather than equipment failure.

In this cyber incident, the conduct of the attacker displayed a great level of familiarity and knowledge about the system. Boden was an insider who worked for a contractor that supplied control system technology to the Maroochy Shire Council. The incident betrays a lack of proper management, technical and operational security controls: no cybersecurity policies, procedures, and defenses were in place. Of particular interest for this work, no proper system monitoring was deployed in the system as a part of a defense-in-depth strategy. As argued by [97], if adequate monitoring systems were deployed, the type of forensics performed by the contractor company and the police might have detected the attack earlier. Since part of the attacker's actions was to stop pumps from running when they should have been, physical domain oriented intrusion detection systems could have detected such incorrect behavior. Moreover, part of the forensics effort performed by the experts was to manually trace back the source of the corrupt messages to a particular pumping station. Alert correlation approaches which automatically link manifestations from both the physical domain (inhibition of pumps) and the cyber domain (corrupt messages) would have assisted investigators in quickly

identifying the source of the incorrect behavior.

1.3.2 Stuxnet

Discovered in June 2010, Stuxnet [39] is a sophisticated worm that is widely suspected of targeting Iran's nuclear facilities, in particular, the Iranian Natanz enrichment plant. The level of Stuxnet's sophistication is manifested through its multiple spreading techniques, its exploitation of four 0day vulnerabilities (application vulnerabilities unknown to the application's developers), its reliance on rootkits (concealment techniques used to hide the malware from anti-viruses and users), its use of two stolen digital certificates to sign its drivers, its modification of system libraries and, finally, its capacity to update itself. The analysis of Stuxnet shows the highly specific nature of its target systems which consist of Siemens SCADA systems with a particular centrifuges layout. Under the assumption that Stuxnet indeed targeted the Iranian nuclear program, Stuxnet might have successfully managed to impact up to 1000 centrifuges at Natanz which amounts to 11% of the total number installed at the time. Due to its high level of sophistication, Stuxnet is suspected to be a joint American/Israeli cyberweapon.

To spread itself, Stuxnet can use multiple methods including infected USB flash drives, network shares, SMB vulnerabilities or Siemens Simatic Step7 projects. Stuxnet also contains safeguards to limit its progression: it can only infect three computers from a given flash drive and is hardcoded to stop spreading itself after June 24, 2012. Through an RPC-based peer-to-peer network connecting it with other instances on a local network, Stuxnet has the ability to update itself to newer versions. Upon infecting a computer, Stuxnet attempts to connect to two command and control (C&C) servers via HTTP. In particular, Stuxnet sends information about the host's OS, computer name, and domain name. It also notifies the server about the presence of particular targeted software such as Siemens Step7 (Siemens specific PLC programming environment) or WinCC (Siemens-specific monitoring software). The C&C server can send Windows binaries to be loaded into the infected machine.

To perform its attack, Stuxnet controls all requests sent to Siemens Simatic PLCs by wrapping a library used to communicate with the PLCs. This allows the malware to install itself on the PLCs and hijack the communication between WinCC and the PLC. As a result, the attacker gains a foothold on the PLCs and can directly issue commands to actuators, effectively blinding any IDS deployed in the supervisory layers.

The code Stuxnet uses to carry attacks defines three sequences of which only two are executed. These two sequences consist of running the centrifuge rotors at too low (2 Hz) or too high frequencies (1410 Hz). The execution of a sequence lasts between 15 to 50 minutes, and successive attacks are separated by about 27 days, which result in stealthy attacks over long periods of time. To conceal its effect on the plant, Stuxnet records and replays to the WinCC monitor program data of the centrifuges' normal operations. As a result, the process operator is unaware of the unusual frequencies at which the centrifuges are running. Detecting this attack would require IDS that: (i) operate between the PLC and the sensors and actuators to avoid being blinded by the replay of recorded data, and (ii) can detect abnormal deviations in the evolution of sensor and actuator states.

Stuxnet is thus an ICS tailored malware with an unprecedented level of sophistication. Its most significant achievement lies undoubtedly in its capacity to leverage Siemens equipment to impact centrifuges using pre-programmed knowledge on the speeds that would cause damage. Beyond protective control measures such as the segmentation and isolation of the various sub-systems comprising the ICS, detecting such attacks requires physical domain IDS that can monitor the state of sensors and actuators, and detect any deviation from normal behavior. In the case of Stuxnet, IDS cannot rely on the monitoring exchanges between the PLCs and the supervisors as they are hijacked by the malware. Ideally, a physical domain IDS would collect data as close as possible to the actuators and sensors, beyond the reach of infected PLCs.

1.3.3 CrashOverride

CrashOverride [138, 32] is the first ever malware specifically targeting electric grids, and the second, after Stuxnet, to be designed with the explicit intent of disrupting physical processes. On December 17th, 2016, the malware successfully impacted an electric substation (that is a system responsible for the transformation of voltage levels, switching operations, and fault protection). Contrary to Stuxnet, CrashOverride is not specific to any particular vendor, relying instead on domain knowledge about electric grids' operations and communication networks. In particular, CrashOverride is extensible with the ability to support additional protocol modules to extend its reach to other geographical regions. Thus, one significant novelty demonstrated by CrashOverride is the move towards malware able to operate in various environments and not confined to specific vendor platforms.

The core of CrashOverride consists of three parts : (i) an initial backdoor, (ii) a loader module, and (iii) several supporting payload modules. The backdoor provides access to the ICS network by connecting to and receiving commands from an external command and control server. Such commands include the creation of new processes, the execution of commands, and the manipulation of users, files and services. The launcher module plays two main roles. It first launches and passes control to the payload modules that manipulate the ICS. Then, it waits for two hours before executing the data wiper which clears registry keys, erases files and kills processes. In particular, the data wiper targets ICS specific configuration files identified through their file extensions. The data wiper's goal is to remove any trace of the attacker's activity in order to evade forensic analysis.

The payload modules are implementations of common electrical grids protocols including IEC 101, IEC 104, IEC 61850, and of OPC DA. These modules read configuration files which define the target (typically a slave device such as a Remote Terminal Unit (RTU) which transmits measurements to a central server) and the action to take. Since there was no observed prior network reconnaissance stage, it is assumed that the configurations are provided by the malware operator. The modules begin by killing the legitimate master process on the victim's host before starting their own process to send data to the slave devices. Then, depending on the configuration file, the module can perform several actions such as (i) enumerating memory addresses corresponding to an RTU

or PLC's inputs/outputs to operate circuit breakers[11], (ii) continuously opening circuit breakers, and (iii) successively opening and closing circuit breakers. Through these payload modules, CrashOverride can de-energize substations potentially leading to outages, or force automated protective operations to isolate a substation due to continuous toggle of breakers between open and closed states.

Among the novelties introduced by the CrashOverride malware is the capacity to engineer process-oriented attacks that can leverage multiple protocols. In terms of intrusion detection, these means that multiple protocol-specific IDS need to be deployed in order to cover all the possible payload modules that CrashOverride is capable of employing. This multiplicity of cyber domain IDS, alongside physical domain IDS that can detect the incorrect handling of breakers, calls for alert correlation approaches that can aggregate alerts from heterogeneous IDS spanning both the cyber and the physical domain.

1.4 Conclusion

In this chapter, we have introduced the class of systems known as ICS. Since these systems from the backbone of many critical infrastructures (energy generation and transmission, transportation systems, water supply, etc.), their correct functioning must be ensured. Following the recent security concerns due to their increased accessibility and convergence with traditional IT systems, we have identified intrusion detection as a pertinent security measure for the automatic identification of security breaches. The differences between an ICS and IT systems mean that a simple transposition of IT security measures into ICS is not always possible, nor desirable. Chief among the specificities of an ICS is the presence of a physical domain in addition to the cyber domain. A discussion of recent attacks on ICS targeting the physical process has motivated the need for physical domain oriented intrusion detection approaches. In the next chapter, we explore the state of the art relative to intrusion detection within ICS.

[11] Electro-mechanical or electronic device designed to protect an electrical circuit from damage due to excessive current

Chapter 2

Related Work

Contents

This chapter presents existing work related to our contributions to intrusion detection (Chapter 3) and alert correlation (Chapter 4) in ICSs. In Section 2.1, we propose a taxonomy of existing intrusion detection approaches in industrial control systems and lay out our positioning with regards to the state of the art. Then, Section 2.2 delves into runtime verification which we identify as a suitable formal verification technique for our intrusion detection approach. Finally, Section 2.3 discusses existing alert correlation solutions similar to our approach.

2.1 Intrusion detection for industrial control systems

This section reviews existing work on intrusion detection for industrial control systems. First, Section 2.1.1 discusses some common taxonomies used to classify ICS-based intru-

sion detection systems. Then, an exploration of ICS-based intrusion detection approaches within the literature is presented in Section 2.1.2.

2.1.1 A taxonomy of intrusion detection approaches in ICS

In Section 1.2.2, we classified IDS approaches following their detection method and data source. In ICS, due to the presence of a physical domain in addition to the cyber domain, IDS approaches can be further characterized by ICS-specific characteristics such as their knowledge of the physical process, the control data (sensor and actuator states) or the control logics executed by the controllers.

The literature contains several examples of ICS-oriented IDS taxonomies [153, 95, 151, 152, 56]. Besides classical distinctions in terms of detection method and data sources, the authors in [153] analyze 9 approaches with respect to: (i) the adequacy of the IDS solution in the context of ICS as reflected by its capacity to handle industrial protocols, its knowledge of the physical process and the hierarchical structure of ICS, (ii) the existence of an explicit threat model, (iii) the distinction between an intrusion and a fault, (iv) the detection time (online or offline), and (v) *integrated security* which refers to the impact on the ICS's availability of a malfunction in the IDS.

Similarly, the authors in [95] extract, through a study of 28 ICS-oriented approaches, the following ICS-specific dimensions: (i) monitoring of the physical process and the laws governing its behavior, (ii) making use of the regularity and predictability of control loops, and (iii) dealing with old and obsolete technologies. The classification proposed in [152] organizes knowledge relative to an industrial system into knowledge about the physical process and knowledge about the control system. Knowledge about the physical process is represented by critical states [19], while knowledge about the control system is subdivided into several aspects: communication, task, resources and control data. Communication deals with protocol-specific aspects such as the vocabulary (syntax of messages) and the grammar (order of message exchanges). It also covers network exchanges at the flow level. A flow is defined as a "set of IP packets passing an observation point in the network during a certain time" [155]; all packets belonging to a flow share common properties. The flow aspect is concerned with information about which nodes communicate with each other and through which protocol these communications are performed. The task aspect covers the scheduling and the state of tasks within the computation nodes of the system. The resources aspect is concerned with the use of memory and computation resources within the computation nodes of the system, as well as the usage of network resources. Finally, control data deals with control-loop characteristics such as the freshness and the interval of allowed values for sensor and actuator states.

Through a synthesis of these different taxonomies, we propose to classify ICS-specific IDS approaches based on the degree of knowledge which the IDS has of the system's interaction with the physical process. Thus, as depicted in Figure 2.1, we distinguish between *cyber domain oriented* intrusion detection approaches that only focus on the cyber aspect of the ICS, and *physical domain oriented* approaches which focus on the physical process.

Among cyber domain oriented approaches, we can further distinguish between IDS

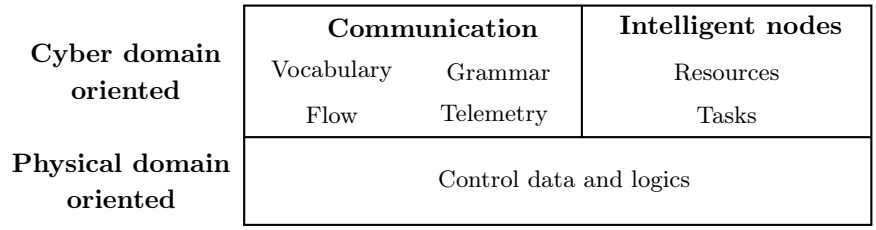


Figure 2.1: Classification of ICS-specific IDS approaches

which focus on the *communication* aspect of the ICS, and IDS which focus on the *intelligent nodes* of the ICS, i.e. nodes which perform some form of computation such as controllers, supervisors or IEDs. This distinction is similar to the one between NIDS and HIDS. The communication aspect covers the communication protocols (vocabulary and grammar), the communication flows (the flow patterns between nodes of the system), and the telemetric profile of the network (statistical meta-information such as the average size or transmission time of network packets). IDSs focusing on intelligent nodes detect suspicious behaviors at the level of the controllers, supervisors, intelligent sensors/actuators, and in general any entity within the system which is able to perform some computation. Here, we are interested in problematics such as memory and computing resources, as well as the scheduling, state, and execution of tasks within intelligent nodes operating under real-time constraints.

For process domain oriented approaches, two aspects are essential: (i) the *control logics* which are executed by the controllers to command the physical process, and (ii) the *control data* which include data sent to actuators and received from sensors, commands issued by supervisors or operators, as well as data exchanged between controllers.

2.1.2 Overview of intrusion detection approaches in ICS

In this section, we provide an overview of intrusion detection approaches in ICS with respect to the classification shown in Figure 2.1. Our objective is to illustrate the strengths and shortcomings of the existing approaches depending on where they lie in the classification.

Compared to misuse-based intrusion detection, anomaly-based approaches have been the focus of research efforts on intrusion detection in ICS. This can be explained by the widespread belief that ICS exhibit relatively stable behaviors due to their fixed topologies and regular communication patterns [21], and by the capacity of anomaly-based approaches to detect novel attacks. Misuse-based intrusion detection efforts have consisted mainly of building signature databases such as those provided by Digital Bond [1]. These signatures can detect denial of service attacks (for instance by rebooting a PLC or sending large size requests), reconnaissance attempts (for instance by scanning available operations and registers on a PLC), and exploits targeting known vulnerabilities such as buffer overflows. The necessity for expert knowledge to write pertinent rules combined

[1] www.digitalbond.com

with the need to constantly keep signature databases updated have further motivated researchers to focus on anomaly-based approaches. In the remainder of this section and thesis, we thus focus mainly on anomaly-based approaches.

A) Cyber domain: communication protocol based approaches

Among IDS that focus on the communication aspect of the ICS, protocol-oriented approaches [24, 85, 48, 68, 69, 23, 148, 33, 54, 149] are interested in the vocabulary and grammar of industrial protocols. For instance, the authors in [24] focus on the Modbus protocol and explore an anomaly-based approach which constrains the values taken by the fields (function code, protocol identifier, etc.) in a Modbus message. Similarly, [85] develop an analyzer for the DNP3 protocol which checks dependencies: (i) within a single DNP3 message such as field lengths, and (ii) between different DNP3 messages such as the occurrence of a response after each request.

While the dependencies on the protocol grammar in [24] and [85] is limited to simple request-response constraints, the works of [48, 68, 69, 23, 148] go farther by inferring an automaton model of the message exchanges. This family of work can be classified in two groups: (i) works that rely on deterministic models [48, 68, 69], and (ii) works that rely on probabilistic models [23, 148]. The authors in [48] explore the possibility of modeling Modbus traffic between an HMI and a PLC (also called an HMI-PLC *channel*) by automatically learning a finite state automaton which models the ordering of messages under the hypothesis that such communications are highly periodic. The aim of the automaton is to capture the re-occurring sequence of control messages, called a *cycle*, which is exchanged on the HMI-PLC channel. In [68], the approach is extended to more complex protocols such as Siemens S7 [68]. However, the evaluation of the approach using data from real plants [23] shows that the periodicity hypothesis of HMI-PLC Modbus traffic is not totally satisfied and that several cycles can co-exist on the same channel. Kleinmann and Wool [68, 70] mention two reasons for this behavior: (i) the presence of a multi-threaded HMI application where each thread scans a separate set of control registers, and (ii) the possibility for an HMI to "subscribe" to some control registers on the PLC and to asynchronously receive notifications about the registers' states. In both cases, the occurrence of several cycles within a channel can lead to a high false alarm rate.

In response to the multiplicity of cycles, Kleinmann and Wool [69] propose to use several automata, each automaton dedicated to a cycle. A demultiplexer is used to select the appropriate automaton for each new message symbol. Yet, the proposed solution relies on a priori knowledge of the number of cycles within the traffic, as well as the use of distinct message symbols for each automaton. All in all, we observe that deterministic approaches assume too strong hypotheses regarding the periodicity and regularity of traffic. For instance, the presence of network delays or operators' interventions can lead to false positives [23].

In reaction to the issues observed in deterministic approaches [48, 68, 69], the works in [23, 148] move towards probabilistic models in the hope to deal with some uncertainties in ICS traffic. In [23], the authors use Markov chains to model the message exchanges

between control servers and PLCs. Each state of the model represents a message symbol, and transitions refer to the sequencing of message symbols. Alerts are raised either due to unknown states/transitions or to states/transitions that occur with an anomalous frequency.

Figure 2.2 shows an example of such an automaton learned from Modbus traffic between a supervisor and a PLC. Labels associated with states refer to types of message characterized by the Modbus function code (FC 5 for write commands, FC 1 and 2 for reading commands) and the accessed PLC addresses (starting address and number of variables accessed). The model shows a group of transitions (drawn in thick) with high probabilities linking in a cycle the states s_0, s_1, and s_2. These states correspond to Modbus monitoring traffic (Modbus read operations). Thus, monitoring data constitutes the majority of the traffic between the supervisor and the PLC. The strong cycle is broken by transitions with low probabilities to the states s_3, s_4, s_5, and s_6. These states correspond to manual operations performed by process operators during the recording of the training traces.

Similarly, [148] model message sequences through probabilistic suffix trees (PST). In comparison with the approach in [23] which conditions the occurrence of a message with respect only to the previous message (Markov assumption), the approach by [148] allows more flexibility by taking into account a variable number of previous messages.

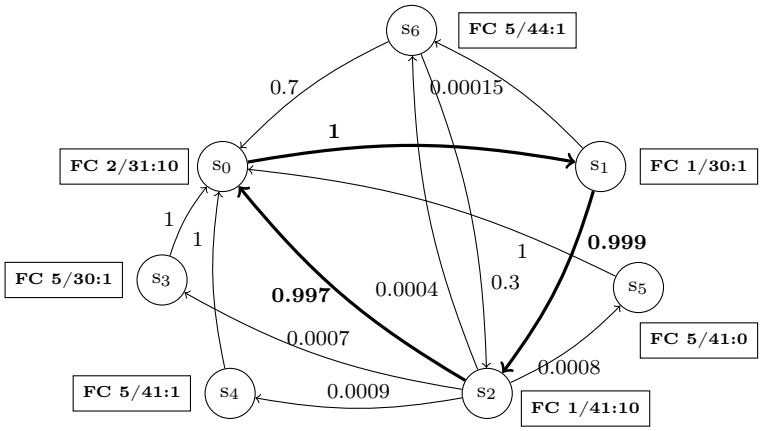


Figure 2.2: Example of a probabilistic automaton model of a supervisor-PLC channel traffic using the approach in [23]

However, as in the case of deterministic approaches [48, 68, 69], the evaluation of probabilistic approaches show that they are effective only when the sequence is strongly periodic with a simple periodicity. Moreover, a common thread running through all protocol-oriented works is the need to ignore parts of the messages in order to generate abstract symbols that allow learning tractable models of the traffic. For instance, to exhibit the periodicity required in [48] and [68], the authors ignore the data carried by the messages and only cater to the type of commands. A similar operation, although to a lesser degree, is performed in the works of [23, 148]. In Figure 2.2, the monitoring

states s_0, s_1, s_2 do not provide any information about the actual state of the physical process since the values returned by the read commands are ignored to keep the model tractable. Such an abstraction operation leads to inaccurate models that run the risk of missing attacks since the view afforded by these models is coarser than the view at the level of the physical process.

Another issue of protocol-based approaches is the lack of information provided by the alerts with regard to the behavior at the level of the physical process. For instance, an unknown state alert raised by the model in Figure 2.2 would boil down to previously unseen access to a PLC variable. Since no context is given as to the state of the physical process when the variable is accessed, the process operator faces difficulties in identifying whether the alert corresponds to a false or true positive.

Dataset	Internal Hosts	Host Pairs	Whitelist
water1	51	58 (1.1)	81 (1.6)
water2-control	22	40 (1.8)	542 (24.6)
water2-field	14	20 (1.4)	23 (1.6)
electric-gas	388	542 (1.4)	1188 (3.1)
loc6	93	23 322 (250.8)	26 759 (287.7)
uni	22 685	56 425 836 (2487.4)	141 744 206 (6248.4)

Figure 2.3: Flow whitelist statistics for a flow-based intrusion detection approach[6]

B) Cyber domain: flow-based approaches

Flow-based approaches [5, 24, 52, 51] focus on the detection of irregular flows within the ICS. The authors in [5] present an approach based on a whitelist of network flows. Each flow is characterized using four properties: (i) the client IP address, (ii) the server's IP address, (iii) the server-side transport port number, and (iv) the transport protocol (TCP or UDP). To justify the tractability of such a whitelist, the authors emit strong hypotheses on the stability of network exchanges within ICS. In the same vein as [5], [52, 51] discuss the possibility of using configuration files available in ICS to extract information pertinent to the detection of unknown flows. In particular, the authors focus on Substation Configuration Description (SCD) files used in IEC 61850 substations and which contain useful information such as the subnetwork to which belongs an IED, its network interface configuration and parameters (IP address, multicast MAC address, IP gateway, etc.).

Figure 2.3 provides statistics about flow whitelists as extracted by [5] on datasets from both industrial networks (*water1*, *water2-control*, *water2-field*, and *electric-gas*) and IT networks (*loc6* and *uni*). The table provides, for each dataset, the set of internal hosts, the number of host pairs in the whitelist, and the number of flows in the whitelist. The table also shows the ratio of host pairs to internal hosts and the ratio of flows to the number of internal hosts. For instance, dataset *water1* contains 51 hosts and its

constructed whitelist contains 58 host pairs (ratio of 1.1) and 81 flows (ratio of 1.6). By comparing the statistics of industrial networks with those of IT networks, we observe that the number of host pairs and of flows generated by IT networks is significantly higher. By contrast, industrial networks are characterized by a limited number of communicating pairs compared to the number of possible pairs.

While the above example shows that network flow whitelists can be tractable in ICS, the presence of mechanisms such as dynamic port allocation (DPA), a technique used with RPC to dynamically decide on ports, can lead to an increase in the size of the whitelist as well as the occurrence of false positives [5]. More importantly, flow-based approaches are limited in terms of the threat model. By construction, such approaches are limited to attacks whose manifestations can appear at the level of flows such as reconnaissance and denial of service attacks [5]. Moreover, an attack can correspond to a legitimate flow leading to false negatives. For instance, an attack where malicious process commands are issued from a supervisor to alter process variables in a PLC's memory can appear as a legitimate flow if the attacker uses the adequate protocols and if the supervisor is otherwise used by process operators to intervene on the process. Finally, an unknown flow does not provide enough information for a security operator to understand the impact of the manifestation on the ICS.

C) Cyber domain: telemetry based approaches

Telemetry oriented approaches [118, 146, 6, 86] focus on building a base profile of network exchanges using statistical measures or classification models. For instance, the authors in [118] assess the performance of several classifiers such as naive Bayesian models, decision trees or logistic regression algorithms in detecting attacks through features such as the number of lost packets from both clients and servers and the retransmission time of lost packets. In this anomaly-based approach, the main assumption is that an attacker's location or software/hardware setup can lead to significant deviations in the network measures. The approach in [146] follows a similar idea, albeit with a more diverse set of measures including host-based indicators such as the number of used processors, the failure rate in connection attempts. In [6], the authors use a spectral representation of flows between control servers and PLCs to detect flows with unusual frequencies.

In general, telemetric approaches are similar to flow-based approaches in that they target a limited threat model. As a result, most approaches focus on attacks that lead to a significant observable deviation in the frequency of flows or of statistical measures such as denial of service [118, 146, 6] or reconnaissance attacks [6]. Thus, a telemetric approach would fail against more subtle and advanced attacks such as process-oriented attacks that only use few commands to put the physical process in a critical state.

D) Cyber domain: intelligent nodes based approaches

Compared to communication-based approaches, intelligent nodes based approaches [154, 124, 121, 13] are relatively scarce. This can be explained by the specificity of the components in an ICS, the limited memory and computing resources, as well as instrumentation

difficulties. The authors in [154] develop an intrusion detection approach based on the analysis of the execution time of tasks in real-time devices. Here, the main target is the detection of code injection attacks which can lead to the execution of unauthorized instructions. In effect, the worst-case execution time (WCET) is estimated from a static analysis of portions of the source code. Then, at runtime, the effective execution time is compared with the WCET bounds and an alert is raised when a task's execution time goes beyond the WCET. The study discusses several instrumentation strategies which allow monitoring of the execution time of tasks without compromising the real-time performance.

Through a different approach, the authors in [124] develop an IDS which models the evolution of variables within PLCs' memories. In effect, the authors estimate the conditional probabilities of the next value of a memory variable given the actual state of the memory. During detection, the IDS raises an alert if the probability of a new value being assigned to a variable is zero. The main assumption of this approach is the regularity of the evolution of the continuous and discrete variables in the PLCs' memories.

All in all, intelligent nodes based approaches remain relatively scarce in comparison with communication-based approaches. This difference can be explained by the difficulties in instrumenting and monitoring the resource-constrained intelligent nodes within an ICS without affecting the real-time performance requirements of control loops [154]. Communication-based approaches are comparatively easier to deploy passively through taps and mirror switch ports without affecting the ICS.

E) Physical domain oriented approaches

As discussed above, flow-based and telemetric-based approaches are ineffective against advanced attacks that induce incorrect behaviors at the level of the physical process without generating overtly abnormal behavior in the cyber domain. And while some protocol-based approaches have attempted to model the sequence of network messages between supervisors and controllers to detect such attacks [23, 148, 48], the abstraction required to keep the models tractable might lead to false negatives. In all the approaches with poor knowledge of the physical process, understanding the meaning of an alert with respect to the behavior of the physical process is difficult. Based on these observations, some IDS approaches incorporate more knowledge about the physical process [55, 35, 20, 42, 96, 112, 111, 142, 40].

Instead of monitoring sequences of network messages, process-oriented intrusion detection approaches generally elect to monitor physical process and control loop related observables such as the evolution of actuator/sensor states or of setpoints. Within this category, the main difference between the approaches pertains to (i) the type of variables that are monitored (continuous or discrete), (ii) the expressivity of the base profile models, and (iii) the cost of specifying the base profile models.

For instance, [55] automatically classify variables involved in a physical process into three types: (i) continuous variables, (ii) discrete variables which can take a finite set of values, (iii) constants. Continuous variables typically represent sensor measures, discrete

$$(\text{PLC}[10.0.0.10 : 502].\text{HR}[1] < 1000 \wedge \text{PLC}[10.0.0.22 : 502].\text{IR}[1] > 99) \rightarrow \text{Alert} : 4$$

Figure 2.4: Example of a critical state-based detection rule[19]

variables represent monitoring information (alarms, events) or a control program's state (clock, counter, etc.), and constants refer to configuration parameters or setpoints. The authors associate a model for each variable depending on its type in order to predict its next value. Thus, continuous variables are modeled through an autoregressive model and by specifying their maximal and minimal bounds. Discrete variables and constants are represented by their set of observed values. The IDS raises an alert when variables manifest abnormal behavior. For continuous variables, this can be manifested by a sudden change in the values or the occurrence of values outside the bounds. For discrete variables and constants, abnormal behavior corresponds to previously unseen values. Similarly, [35] differentiate three main types of PLC registers used by Modbus: (i) measure variables, (ii) counter variables, and (iii) constants. Here, the base profile of variables is constructed through the statistical variance, dispersion indices, or monotonicity. Both [55, 35] reduce the cost of specifying the base profiles by automatically determining the type of variables.

In comparison with [55, 35], the works of [20, 42, 96] differ in that they start from manual specifications about forbidden states or allowed behavior within the physical process. Such specifications are often expressed in terms of interval values on measure vectors. The authors in [20, 19, 42] present an approach which is based on the a priori specification of the critical states within the physical process. The state of the physical process corresponds to the values taken by actuators and sensors. An alert is raised when the current state reaches or approaches a forbidden state. In the same vein, [96] develops an IDS based on manually specified behavioral rules which circumscribe the allowed behavior of the sensors and actuators. The heart of the approach consists in transforming the behavior rules into a probabilistic finite state automaton representation where states are partitioned into secure states, warning states, and non-secure states. The transition probabilities are computed for different hypotheses of noise and attacker models. The detection of an attack is based on a compliance degree which reflects the tendency of the system to behave securely. A device behaves securely when its state, as represented by the state of actuators and sensors, remains in or near secure states.

An example of a detection rule based on critical states is shown in Figure 2.4, written using a custom language called ISML [19]. The left side of the rule expresses constraints on a holding register (HR) in the PLC at address 10.10.0.10, and on an input register (IR) in the PLC at address 10.10.0.22. These registers correspond respectively to a continuous actuator (turbine rotation speed) and a continuous sensor (temperature sensor). The semantics of the rule is that an alert with risk level 4 is raised if the turbine rotation speed is less than 1000 and the temperature is above 99.

Based on the previous example, we can make the following remarks about the expressivity and complexity of such models. In general, ISML expresses constraints as a simple

conjunction of assignments to PLC registers with no notion of temporal order. Thus, the model does not allow expressing rules over *events*, i.e. changes in the state of the actuators or sensors, or over *temporal* behaviors involving states and events. For instance, one cannot express a rule that raises an alert if a valve is opened more than 10 seconds after shutting down a motor. Expressing such rules requires more expressive models that reason on trajectories instead of states. Moreover, the approach assumes that the set of critical states can be manually determined in advance. While such an assumption can be valid for small systems, manually determining critical states for large systems can be daunting and involves the risk of missing some critical states. Expressivity limits apply to many of the approaches in this category [20, 42, 96]. As a result, recent approaches have begun exploring more expressive formalisms such as temporal logics [128, 40] which can be used to specify temporal patterns over the behavior of the system. Similarly, the authors in [2] argue for the centrality of quantitative time within cyber-physical systems and derive security constraints which incorporate time in addition to states and events.

While the previous approaches assume that access to actuator and sensor states is possible at low layers of the system, the authors in [53] follow a radically different approach by developing a semi-automatic method to detect malicious actions based on a semi-automated SCADA log processing approach. The authors propose to rely on two methodologies originating from the field of industrial risk analysis: PHEA (Predictive Human Error Analysis) and HAZOP(HAZard and OPerability study). PHEA is used to specify the set of possible actions on a SCADA system. These actions serve as input to the HAZOP analysis, which is a qualitative method that identifies risky situations based on the combination of keywords relative to the process, and of guidewords that specify possible process deviations. Risky situations are then classified based on the severity of their consequences, and the most critical situations are detected based on the processing of logs. Under the hypothesis that SCADA logs are relatively stable (limited changes to equipments, existence of a finite set of repeated actions), the analysis is based on the identification of least frequent patterns of system behavior based on a set of attributes including the timestamp, the type of action, the user initiating the action, and the action's location. Least frequent patterns are expected to reflect anomalous behaviors.

As discussed in Section 1.2.1, the notion of *measures integrity* is essential in ICS. A category of intrusion detection approaches is specifically interested in false measures injection attacks [142, 22, 132]. For instance, the authors in [22] stress the necessity to develop algorithms able to detect compromised measures where the attacker alters the measures coming from sensors. The authors propose to use knowledge about the command laws and the physical process models (state and output equations) in order to predict the expected measures with respect to a sequence of commands. An attack is detected by comparing the actual and the predicted measures. The detection is based on sequential detection algorithms which determine, given a sequence of observations and a threshold value which is set to respect a false alarm rate, whether the system is under attack. Three attack types are considered: (i) surge attacks where the attacker maximizes the impact as soon as possible, (ii) bias attacks where the attacker alters the measure vectors by a constant value at each sample, and (iii) geometric attacks which

combine both surge and bias attacks. The authors conclude that geometric attacks are the most successful attack strategy while surge attacks do not cause significant damage to the process.

Approaches in this category are often geared towards specific ICS applications. For instance, [132] propose an approach adapted to the case of power systems. Here, detection is based on a temporal series of discrepancies in the measured frequencies and power. [72] explore, in a detection approach geared towards smart grids, to use a set of rules extracted from a hybrid automaton model of the system. The automaton characterizes the behavior of the system in terms of the evolution of continuous (current intensity) and discrete (state of breakers) values. The authors in [44] adopt an anomaly-based approach based on neural networks applied to the control of a water tank. The features that are considered are: (i) the level of the tank, (ii) the frequency of the responses, (iii) the operation mode of the controllers (manual or automatic) and iv) the state of the pump.

The necessity for process-oriented intrusion detection is motivated by the shortcomings of traditional intrusion detection approaches in detecting attacks targeting the physical process as demonstrated by some major recent security incidents (Stuxnet [39], CrashOverride [138]). In contrast to the approaches that rely solely on cyber domain observations to detect attacks targeting the physical process [23, 148, 48], process-oriented approaches directly monitor the evolution of process variables [20, 55, 35].

However, some approaches [94, 19] only cover a limited threat model due to the low expressivity of their models. For instance, the approach developed in [94] can only detect global invariants valid over all the states of the physical process (ex. the property that a temperature should never exceed a maximum threshold anytime during the execution of the process). However, some attacks can violate invariants that are local to specific states of the physical process (ex. a valve which can only be opened during a specific state of the physical process). On the other hand, approaches with more expressive formalisms involve costly detection models [128, 149] in terms of time and required expert knowledge. As a result, research is still required to develop highly automated and cost-effective process-oriented approaches targeting a large threat model. In Chapter 3, we propose an intrusion detection approach which uses a sufficiently expressive formalism to cover the sequential dynamics of an ICS. Our approach automatically builds the detection models through an inference phase in order to alleviate the operators from the burden of manually specifying the models.

F) Approaches combining different aspects

Finally, a number of works [14, 15, 113, 152] cover multiple aspects of the above taxonomy. Such a wide coverage is motivated by the need to detect sophisticated attacks, identify accidental deviations, and reduce false positives. Following a formal approach, [14, 15] are interested in intrusion detection within advanced metering infrastructures (AMIs). An AMI automatically collects data from energy metering devices (electricity, gas, water, etc.) for energy consumption analysis and billing purposes. The authors develop an architecture including a set of intrusion detection and alert aggregation agents. The intrusion detection agents span the communication dimension (structure, protocol

grammar, telemetry) as well as the intelligent nodes dimension (constraints on objects access and resource use). The specifications identified for the different dimensions are fed to a formal verification environment including a proof assistant. This environment includes also the specification of the global security policy which needs to be guaranteed. The goal is to prove that there exists no network trace which can violate the security policy without being detected by the intrusion detection agents.

The authors in [152] introduce an intrusion detection approach combining knowledge about the physical process and the cyber domain. With respect to the communication aspect, the authors focus on grammar, structure, and telemetry. Knowledge about the physical process is represented by critical states as in [19]. The authors develop a statistical anomaly alert classification approach based on a hidden Markov model (HMM). Alerts from different anomaly-based IDS are reduced to a tuple of binary values where each position in the tuple is associated with an IDS. The HMM estimates whether a tuple corresponds to an attack, a fault, or a normal behavior.

While the above approaches explore the joint use of different types of intrusion detection approaches [113, 152], the treatment of alerts remains simplistic, leading to an increased awareness in the need for *alert correlation* [14]. In IT systems, alert correlation [139, 29, 140] is a set of techniques used to eliminate redundant alerts, reduce the number of false alerts, and reconstruct attack scenarios. Yet, alert correlation approaches [17] in ICS remain scarce. We provide a more in-depth coverage of the applicability of classical alert correlation approaches to ICS in Section 2.3. Here, we briefly mention some ICS-specific approaches to alert correlation.

The authors in [17] develop a solution to detect, correlate and visualize multi-steps attacks. In such an attack, the attacker needs to cross multiple network zones to reach the critical zones of the ICS. The detection combines misuse-based approaches (Snort with Digital Bond signatures), and anomaly-based approaches (Bayesian sensors) which detect reconnaissance and denial of service attacks. Correlation and visualization are performed through the ArcSight Security Information and Event Management (SIEM) system. To deal with the important number of alerts that can be returned by the IDS, the alerts are classified by incident classes. A priority level is assigned to each class depending on its consequences on the security objectives of the ICS. Thus, incident classes that affect availability are given higher priority relative to integrity and confidentiality. In parallel, a criticality level is associated with each component of the ICS based on its type and deployment zone. For instance, controller components and field-level components receive maximal criticality. Correlation is then performed through an algorithm which takes into account the progression in the criticality of the alerts raised by the IDS. As a result, the correlated alerts represent the progression of the attacker towards increasingly critical parts of the ICS.

In summary, research on the combined use of multiple IDS for the detection of attacks is still limited. The works in [14, 15], although covering multiple aspects, does not involve knowledge about the physical process. A similar observation can be made for the correlation approach in [17]. And while the solution in [152] includes knowledge of the physical process in terms of critical states [19], the reduction performed on the

alerts discards information which would otherwise help the operator in understanding the suspicious manifestations (impacted actuators, IP addresses, etc.). Thus, an important and still unresolved issue facing approaches covering multiple aspects is the need to associate suspicious manifestations coming from both the cyber and the physical domains despite their disparity in terms of attributes (actuator/sensor states and events in the physical domain, protocol-based attributes in the cyber domain). In Chapter 4, we propose to tackle this issue through an alert aggregation approach that caters for the heterogeneous nature of ICS domains.

2.2 Runtime verification

In this section, we first explore various formal verification techniques and then focus on runtime verification which we identify as a suitable technique for our intrusion detection approach presented in Chapter 3.

2.2.1 Motivation

As we discuss in Section 3.1, our threat model focuses on process-oriented attacks, i.e attacks which induce incorrect behavior at the level of the physical process. Correction is defined as the occurrence or non-occurrence of temporal patterns, also called *specifications* [90]. For instance, a specification can define a correct trajectory as a trajectory where a forbidden state is never reached (i.e overflowing a tank, going over a certain temperature threshold, etc). In terms of intrusion detection, our objective is to *verify*, during execution, whether the physical process deviates from correct sequential behaviors.

IEEE [41] defines *verification* as *the set of techniques which aim at showing that a system satisfies its specifications*. Examples of such techniques include *model checking*, *theorem proving*, *testing*, and *runtime verification*. The selection of a technique depends, among other factors, on the expected exhaustivity of the verification (coverage of all possible behaviors of the system), the automation of the verification procedure, the complexity of the systems under study, and the stage where verification is performed (development, deployment, etc.).

Intrusion detection is an *a posteriori* security measure used mainly during the system's deployment where only the *current behavior* of the system is accessible. Thus, we do not aim at an exhaustive coverage of all possible behaviors of the system; the IDS should only be able to say, for any current behavior, whether it is correct or not.

Thus, we frame our intrusion detection task as the *automatic verification of a set of correctness properties on the sequential behavior of the physical process during its execution*. We now briefly discuss each of the main verification techniques and argue that *runtime verification* is the most suitable technique to serve as the basis for our intrusion detection task.

Model checking The objective of model checking [25] is to verify *automatically* that all trajectories of a model satisfy a specification. Here, the model is a mathematically

precise description of the behavior of the system under study. Model checking is thus an automatic and exhaustive approach to check whether all possible system scenarios, *as captured by the model*, satisfy some desired properties. Due to the exhaustivity of the verification, model checking is mainly applicable to finite state systems whose trajectories can be enumerated. This imposes limitations on the size and complexity of the systems under study. Moreover, the verification procedure happens mainly during the conception and development stages. This approach is thus not suitable for verification during deployment.

Theorem proving Theorem proving [114] is a *deductive* verification technique which, starting from a mathematical representation of the system under study, attempts to *deduce* a theorem showing that a desired property is satisfied. To construct a proof of the theorem, software tools, known as *proof assistants*, generate intermediate proof steps. Proof assistants generally require human assistance to reduce the proof search. Thus, this technique is labor-intensive and requires a high level of expertise. Moreover, this approach is geared towards the conception and development stages of a system and is thus not adapted to the verification during the system's deployment.

Testing When a model description of the system is unavailable (ex. in the case of proprietary models), model checking and theorem proving techniques cannot be applied. In this case, an alternative approach, called *testing* [110], consists in generating and feeding inputs to the system and then checking for an appropriate reaction. Testing can be performed either with a total knowledge of the system (*white-box testing*), with no knowledge of the system (*black-box testing*) or with intermediate knowledge (*gray-box testing*). Testing suffers from its lack of exhaustivity and the difficulty to test deployed safety-critical systems where bad reactions can lead to significant damages.

Runtime verification Following [82], runtime verification is defined as the "*discipline of computer science that deals with the study, development, and application of those verification techniques that allow checking whether a* ***run*** *of a system under scrutiny satisfies or violates a given correctness property*". In practice, verification is performed using a *monitor* derived automatically from the property to be verified. In an online setting (as opposed to processing recorded behavior), the monitor follows the **current behavior (i.e run)** and, at any instant, delivers a verdict on whether the property is satisfied or violated. Thus, runtime verification differs significantly from the above techniques in that it is geared towards verification during deployment.

Discussion Among all the popular verification approaches sketched above, runtime verification is the only one allowing verification during the deployment of the system under study. Moreover, since it does not target exhaustivity, its verification task is significantly less costly in terms of computation and human involvement compared to model checking and theorem proving. Thus, we adopt runtime verification as the basis for our

intrusion detection approach. In the following sections, we explore popular specification languages used in runtime verification and discuss the synthesis of runtime monitors.

2.2.2 Specification languages for runtime verification

As discussed above, specifications correspond to properties which the system needs to satisfy. Usually, such specifications are expressed in terms of natural language (ex. English) and can thus fall prey to ambiguous interpretation. To resolve this ambiguity, several *specification languages* have been devised. These languages formally define the syntax (set of well-formed specifications) and semantics (interpretation of a well-formed formula) of specifications.

A) Characteristics of specification languages

Specification languages can be broadly classified based on the following aspects [7]:

- **Executable/Declarative** A distinction can be made between *executable* specification languages such as state machines and *declarative* specification languages such as temporal logics. Executable specification languages produce low-level specifications that can use more straightforward monitoring algorithms. On the other hand, declarative specification languages capture properties at a higher level of abstraction. Declarative languages require a procedure to generate executable objects (monitors) which can be used to verify the properties.
- **Finite/Infinite** When verifying a system at runtime, the observations are necessarily (ever-increasing) *finite* traces. However, the semantics of some specification languages are defined over *infinite* traces (ex. temporal logics). Thus, in order for such languages to be used in runtime verification, a mapping is performed from infinite to finite semantics.
- **Qualitative/Quantitative time** Some specification languages can only denote the relative order (*qualitative*) of states/events on a trace, while others include a notion of *quantitative* time distance between states/events. For instance, specification languages supporting a quantitative notion of time can check whether event A happens less than x seconds after event B, whereas a qualitative notion of time would only express whether A happens before or after B.
- **Data** While classical specification languages consider states/events as atomic symbols, recent research has explored more expressive specification languages that can support structures containing data variables [131] (ex. records in a relational database, data variables in sequential programs, etc.).

Using runtime verification for the purpose of intrusion detection against the threat model discussed in Section 3.1 forces some choices with respect to the aforementioned aspects. Firstly, since runtime verification is performed on observations which are (ever-increasing) finite traces, we require specification languages with finite semantics. Also,

because our threat model covers the sequential behavior of the physical process, we require specification languages that can express both qualitative and quantitative time constraints. We do not require however the full power of specification languages supporting data variables as we only consider constraints over logical states and events. Finally, for declarative specification languages, a monitoring procedure must exist which allows the monitoring of the specifications.

There is a significant amount of specification languages discussed in the literature [7]. Common languages include temporal logics, regular and non-regular languages, automata and rule systems. In this work, we focus on the most studied family of specification languages, namely temporal logics. Due to their widespread use, this family includes variants which cover nearly all the possible configurations of the aspects discussed above.

B) Temporal Logic

Temporal logic is the most common family of specification languages for runtime verification. Temporal logics are declarative specification languages, usually interpreted over infinite traces with adaptations to finite traces. The family includes variants supporting both qualitative and quantitative time. Temporal logic has also been extended to support data variables [131] and description logics [89]. We focus in particular on two common variants which together fulfill the needs of our intrusion detection task: *Linear Temporal Logic (LTL)* and *Metric Temporal Logic (MTL)*. Through these specification languages, we can express and monitor specifications over the sequential behavior of an ICS.

Linear temporal logic [117] augments propositional logic with *temporal operators* able to express ordering relationships. LTL formulae are interpreted over infinite sequences where each position is associated with a set of true boolean variables. An example of a temporal operator is the *always* operator, denoted $\Box$. For any propositional formula p, the LTL formula $\Box p$ is satisfied at a position of a trace if p is satisfied in every subsequent position of the trace. For instance, given the sequence $\sigma : \{a\}.\{a,b\}.\{a\}\ldots$ with propositional variables a and b, the formula $\Box a$ is satisfied at the first position since a is true at every position of the sequence.

Another temporal logic operator is the *eventually* operator, denoted $\Diamond$. For any propositional formula p, the LTL formula $\Diamond p$ is satisfied at a position of a trace if there exists a subsequent position of the trace where p is true. For instance, given the sequence σ above, the formula $\Diamond b$ is true at the first position since b eventually becomes true at the second position of the sequence.

In addition to the above temporal operators, LTL also includes the *next* operator, denoted X, which asserts that a formula is true at the next position of the trace. For instance, Xb is true at the beginning of σ since b becomes true at the next position. Finally, the *Until* operator, denoted $\mathcal{U}$, takes two formulae as input. The LTL formula $p\mathcal{U}q$ asserts that p is true until q becomes true. Thus, $a\mathcal{U}b$ is true at the first position of σ.

We now give a formal presentation of LTL's syntax and semantics. Let AP be a set of atomic propositions. AP might, for instance, contain propositions about the state of actuators and sensors. The syntax of LTL over the alphabet $\Sigma = 2^{AP}$, which we write

$LTL(\Sigma)$, is defined as follows:

$$\varphi :: p \mid \neg\varphi \mid \ \varphi \vee \varphi \mid \varphi U \varphi \mid X\varphi,\ p \in AP$$

We define Σ^ω as the set of infinite sequences over Σ. Let $\varphi, \varphi_1, \varphi_2 \in LTL(\Sigma)$ be LTL formulae, $i \in \mathbb{N}$ a position, and $w(i)$ the ith element of the infinite sequence $w \in \Sigma^\omega$. LTL formulae can be inductively interpreted over elements in Σ^ω as follows:

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

We also define $\Diamond\varphi \equiv trueU\varphi$ and $\Box\varphi \equiv \neg\Diamond\neg\varphi$. Here, $\neg$ and $\vee$ are, respectively, the negation and logical OR operators. The remaining logical operators $(\wedge, \Rightarrow, \Leftrightarrow)$ can be derived as usual.

In sequential control systems, we are often interested in expressing properties involving *events* such as *rising* $(\uparrow)$ or *falling* $(\downarrow)$ edges. Such events can be expressed in LTL as follows [119]:

$$a^\uparrow \equiv \neg a \wedge Xa \qquad a^\downarrow \equiv a \wedge X\neg a$$

Example 2.1. *(LTL formulae)* The constraint that valves $VP1$ and $VP2$ are never simultaneously opened can be expressed as the LTL formula: $\Box\neg(\mathrm{VP1}_{\mathrm{open}} \wedge \mathrm{VP2}_{\mathrm{open}})$ where $\mathrm{VP1}_{\mathrm{open}}$ and $\mathrm{VP2}_{\mathrm{open}}$ are propositional variables referring to the open state of valves VP1 and VP2.

While temporal operators in LTL only allow specifying qualitative temporal relationships, ***Metric Temporal Logic (MTL)*** extends LTL to specify quantitative temporal properties. Various types of MTL semantics have been explored in the literature[9]. Since we observe the state of actuators and sensors by taking samples at discrete times, we focus on the so-called *point-based* semantics of MTL over a discrete time domain. MTL extends LTL's operators with *intervals*. An interval refers to a range of discrete time points. For instance, the MTL formula $\Box_{[1,5]}p$ asserts that p should hold between 1 and 5 time points from now. Contrary to LTL, MTL formulae are interpreted over *timed sequences* which associate every time point with a set of true propositional variables. Timed sequences are of the form $\rho = (\tau_0, E_0).(\tau_1, E_1)\ldots$ where $\tau_i \in \mathbb{R}^+$ and (τ_i) is strictly increasing and either finite or diverges to infinity (*non-Zenoness* requirement), and E_i is a set of boolean variables which are true at τ_i.

Formally, the syntax of MTL over the alphabet $\Sigma = 2^{AP}$, which we write $MTL(\Sigma)$, is defined as follows:

$$\varphi :: p \mid \neg\varphi \mid \ \varphi \vee \varphi \mid \varphi U_I \varphi,\ p \in AP$$

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

Given a timed sequence ρ and MTL formula φ, $\rho, i \models \varphi$ is interpreted inductively as follows:

$$\begin{aligned}
\rho, i \models p \in AP &\iff p \in E_i \\
\rho, i \models \neg\varphi &\iff \rho, i \not\models \varphi \\
\rho, i \models \varphi_1 \vee \varphi_2 &\iff \rho, i \vDash \varphi_1 \vee \rho, i \vDash \varphi_2 \\
\rho, i \models \varphi_1 U_I \varphi_2 &\iff \exists j \text{ such that } i < j < |\rho|, \rho, j \models \varphi_2, \tau_j - \tau_j \in I, \\
&\qquad \wedge \forall k \text{ with } i < k < j, \rho, k \models \varphi_1
\end{aligned}$$

As for LTL, we also define $\Diamond_I \varphi \equiv true U_I \varphi$ and $\Box_I \varphi \equiv \neg\Diamond_I \neg\varphi$.

Example 2.2. *(MTL formulae)* The constraint that the valve $VT1$ must be opened at least once each hour can be expressed with the following MTL formula: $\Box(\Diamond_{[0,3600s]} \mathrm{VT1}_{\mathrm{open}})$.

2.2.3 Monitoring techniques

As discussed above, monitors only have access to finite but expanding prefixes. However, LTL and MTL formulae are interpreted over infinite sequences. This mismatch restricts the class of *monitorable* formulae [10]. Monitorability refers to the capacity of a monitor, after any finite number of observations, to still detect the violation/satisfaction of a property after, at most, a finite number of additional observations. Intuitively, this means checking whether the monitor can still provide an evaluation of the current execution while avoiding situations where the monitor would inevitably provide inconclusive results [7]. For instance, consider the following two properties: (i) $\Box\neg(valve_1 \wedge valve_2)$, and (ii) $\Box\Diamond(valve_1)$. Property (i) says that $valve_1$ and $valve_2$ should never be simultaneously active. At any time during the monitoring of this property, it is still possible to reach a point where the property is violated, i.e when we observe a state where $valve_1$ and $valve_2$ are both simultaneously active. In this case, the monitor can confidently declare the violation of the property. On the other hand, property (ii) says that it is always the case that we will reach a position where $valve_1$ is active. Contrary to property (i), the monitor can never hope to reach a point where it is possible to decide whether this property is violated or not. To observe this, note that irrespective of any new input, the monitor still needs to look out for the occurrence in a future position of a state where $valve_1$ is active. Thus, the monitor is always in a state of indecision and this property is deemed not monitorable.

Several characterizations of monitorable properties exist within the literature [38, 81]. Of particular interest in the detection of process-oriented attacks is the subset of monitorable properties called *safety* properties. Informally, a safety property states that "something bad should never happen". The property (i) above is an example of a safety property. Another class of monitorable properties is called *co-safety* properties. Informally, a co-safety property states that "something good happens". For example, the property $\Diamond(valve_1)$ is a co-safety property which says that $valve_1$ is finally open at some point in time. In the remainder of this work, and because we want to monitor properties

for which we have a hope of reaching a definite verdict, we only focus on monitorable properties [38, 81].

Two main methods are used to realize the monitoring functionality: *automata-based* and *rewriting-based* techniques.

Automata-based techniques. Using these techniques, monitors are synthesized as finite state automata from property specifications. For instance, in the case of LTL safety properties, such an automaton recognizes minimal bad prefixes of a safety property. Minimal bad prefixes are finite sequences which cannot be extended to satisfy the safety property, and which do not contain any other bad prefix [27]. If a safety property is violated on an infinite sequence, then it has already been violated on some finite prefix. In this case, a monitor is a finite state automaton which recognizes, as early as possible, such a prefix and reports a violation. Constructing a monitor usually requires translating the LTL formula into a Büchi automaton which accepts all infinite sequences satisfying the formula (see [141] for a formal definition). A nested depth-first-search allows the identification and removal, from the Büchi automaton, of all states which cannot initiate an accepting run. The resulting automaton can then be treated as a finite state automaton where all states are *accepting* and used as a monitor [27]. Similar automata-based techniques have been developed for quantitative specification languages [59].

Rewriting-based techniques. While automata-based techniques transform specifications into automata prior to processing any observation, rewriting-based techniques [57, 58], operate directly on the sequence of states and events. After each processed state or event, the monitored property is rewritten and the new property's validity is tested on the remaining sequence of states and events. This rewriting process continues until the truth value of a newly generated property can be determined. Rewriting-based techniques can also transform the original formula into an equivalent representation which allows for efficient monitoring by, for instance, keeping track of the truth value of subformulae through dynamic programming methods [8]. In practice, automata-based techniques can sometimes be considered as pre-computations of rewriting-based approaches [80, 81]. If the automaton is realized as a transition table, a simple lookup is sufficient to process any observation. However, the size of the transition table can be huge depending on the size of the formula to be monitored. Thus, while rewriting-based techniques are typically easier to implement and may exhibit better memory performance, they may also suffer from worse runtime performance [81].

In selecting between a rewriting-based and an automata-based technique, one essentially needs to strike a memory-time trade-off. For the purposes of intrusion detection, the verdict needs to be reached as soon as possible and runtime performance is critical, thus favoring automata-based techniques. Moreover, according to [12], most common LTL properties result in a monitor of size less than 100 measured in terms of the number of states and transitions. However, on a case-by-case basis and for pathological properties which result in the synthesis of a big automaton, rewriting-based techniques can be adopted instead.

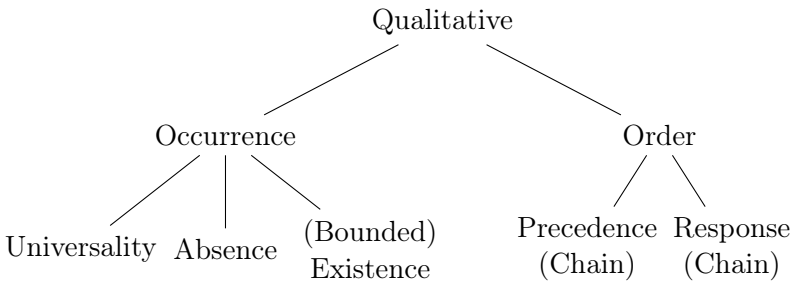


Figure 2.5: Classification of qualitative specification patterns

2.2.4 Specification patterns

While LTL and MTL provide suitable formalisms to characterize temporal constraints pertaining to states and events ordering, expressing specifications directly in terms of formulae can be tedious. As properties grow in complexity, writing accurate and correct formulae becomes a difficult task. Thus, several works [34, 119, 71] have looked at *specification patterns* that capture commonly occurring properties. As argued in [34], only a small proportion of the constraints expressible by any given specification language occur in practice. Thus, a reduced set of reoccurring specifications can be identified through an extensive review of the literature (e.g. requirement documents for appliance and automotive embedded systems [71]), and then generalized to produce specification patterns. Moreover, specification patterns provide a more intuitive description of specifications in terms of natural language. This turns out to be extremely beneficial for our intrusion detection task. Instead of parsing a complex temporal logic formula or a low-level automaton, the security operators can interpret a specification by referring to its specification pattern class which is described in natural language.

For our purposes, specification patterns can be divided into two groups: (i) *qualitative specification patterns* [34, 119] which represent qualitative temporal properties based on a formalism such as LTL, and (ii) *quantitative specification patterns* [71] which include quantitative time constraints using a formalism such as MTL. We now further discuss each group based on the seminal works of Dwyer [34] and Konrad [71].

Qualitative specification patterns Dwyer [34] distinguishes between *occurrence patterns* and *order patterns*. Occurrence patterns include *absence*, *universality* and *existence* patterns. Absence patterns assert that a certain event or state never occurs during the execution of the system. Universality patterns assert that a certain state always holds during the execution of the system. Existence patterns assert that a certain event or state eventually holds at some point during the execution of the system. Order patterns include *precedence* and *response* patterns. Precedence and response patterns express relationships between two events or states where the occurrence of one is a necessary condition for the occurrence of the other.

In order to further restrict the portion of the execution where a specification should

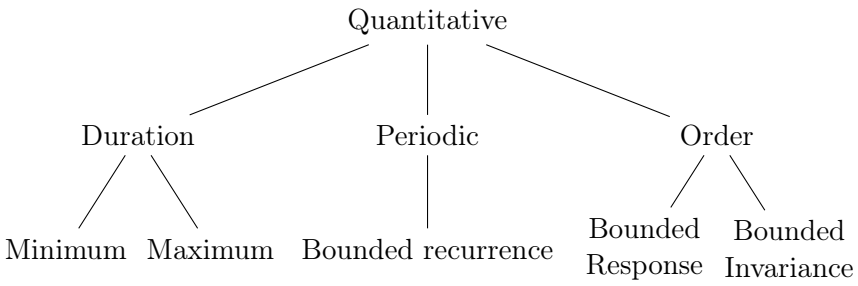


Figure 2.6: Classification of quantitative specification patterns

hold, Dwyer [34] introduces the notion of a *scope*. Five scopes are defined: (i) a *global* scope, (ii) a scope starting *after* an event/state, (iii) a scope ending *before* an event/state, (iv) a scope *between* two events/states, (v) a scope starting *after* a first event/state and lasting *until* the eventual occurrence of a second event/state. All scopes are left-closed and right-open. In practice, scopes allow us to express local constraints on the execution of the physical process.

Quantitative specification patterns Konrad [71] extends the fundamental notions introduced by Dwyer's to the quantitative domain. Three categories of quantitative specification patterns are identified: *duration*, *periodic*, and *order*. Duration patterns describe the minimum or maximum amount of time a state formula has to hold once it becomes true. Periodic patterns describe the amount of time in which a formula has to hold at least once. Finally, order patterns specify either the maximum amount of time that passes between a first state holding true and a second state holding true (*bounded response*) or the minimum amount of time a state must hold true once another state holds true (*bounded invariance*). As in the case of Dwyer's patterns, scopes restrict the portion of the execution trace on which the specifications must hold.

Discussion Qualitative (Figure 2.5) and quantitative (Figure 2.6) specification patterns constitute our starting point in terms of sequential behavior constraints. However, patterns are abstract entities which need to be instantiated on a particular system in order to yield concrete specifications that can be monitored. Our next objective is thus as follows: given a finite set of specification patterns and a finite set of execution traces of the physical process, find all instantiations of the patterns that are valid on the traces. This task is known as *specification mining* and requires special care to avoid generating too many irrelevant instances.

2.2.5 Specification mining

The problem of specification mining can be expressed as follows: *given a finite set of specification patterns and a finite set of execution traces of a system, find all instantiations that are valid on the traces.* Several works have explored this issue based on a

variety of patterns [79, 84, 147, 144, 88, 62]. Most specification mining approaches [144, 88, 147, 84] focus on specific specification patterns or tool-specific templates such as response [147, 144, 88] and alternating patterns [144]. In contrast, the approach in [79] develops a specification mining approach geared towards LTL patterns of arbitrary length and complexity. Similarly, mining of quantitative temporal has also been explored in the literature [62].

Regardless of the specific mining approach adopted, one is usually required, if no a priori information is available, to explore the space of all possible instantiations and test the validity of each instantiation on the traces. While the size of the search space can be significant, recent work [79] has shown that using memoization and selective treatment of the traces can significantly reduce the complexity of the task even when dealing with general LTL formulae. However, as we discuss in Chapter 3, the number of resulting mined specifications can still remain significant, and a large subset of these specifications are often redundant. In terms of intrusion detection, this phenomenon can lead to the generation of too many alerts with poor value to the security operator. In Chapter 3, we propose an approach to mine relevant specifications in the particular context of ICS.

2.3 Alert correlation

In order to link alerts from both the physical domain IDS and the cyber domain IDS, we propose in Chapter 4 an alert correlation approach adapted to the context of ICS. In this section, we introduce the notion of alert correlation and present the individual phases that form the overall correlation process. We explore related work for each of these phases and discuss their adequacy with respect to ICS.

2.3.1 Motivation

The number of alerts and the volume of data generated by IDS in a system can be overwhelming for security operators [105]. Some attacks such as port scans and denial of service attacks tend to generate multiple alerts. Thus, one or more IDS can emit multiple alerts for a single attack. This phenomenon can also lead to redundant alerts which an operator needs to sift through in order to identify pertinent information. Moreover, due to poor IDS signatures in the case of misuse-based IDS, or incorrect base profiles in the case of anomaly-based IDS, a significant amount of alerts can consist of false positives.

To further motivate the need for correlation, this time in the context of ICS, consider the case of process-oriented attacks carried out from the supervisory levels [32]. While physical domain IDS might be able to detect the effects of the attack on the physical domain, they do not provide information about the source of the attack at the supervisory level. On the other hand, such information can be available at the level of cyber domain IDS which in turn lack any visibility about the impact of the attacks on the physical process. Thus, to acquire a global view of the attack, it is necessary to aggregate alerts from both the physical and the cyber domains. Aggregating alerts is one of the core tasks of alert correlation.

Overall, the main objectives of alert correlation approaches are to (i) reduce the number of alerts by removing false positives and redundancies, (ii) group low-level alerts by reconstructing attack scenarios that afford operators a higher-level view and give scores of criticality to alert groups.

2.3.2 Phases of alert correlation

While several alert correlation models exist within the literature [140, 125, 127], the following four-step general architecture can be identified [47]:

A) Pre-treatment

Raw alerts coming from the IDS often require pre-treatment in order to be ready for the correlation process. For instance, alerts coming from different IDS often use different formats and conventions. Thus, a **normalization** step is added to unify the syntax (i.e., the structure of the attributes) [66, 134] and semantics (i.e., the meaning of the attributes) [66, 87, 134] of the alerts to allow for their comparison in the correlation process. For instance, the works in [134, 66] rely on the IDMEF format [2] which defines a unified format that enables syntactic normalization. Semantic normalization usually consists of finding equivalencies between different attributes' values or establishing generic attack categories. As an illustration of semantic normalization, the authors in [134] extend IDMEF with a *taxonomy* attribute that allows specifying the meaning of alerts based on a hierarchy of objects (system, protocol, user, etc.) and some associated conditions or actions on the objects. For instance, an alert on the shutdown of an operating system would be reported as a *shutdown* action on the *system.os* object.

In general, alert normalization is either implicitly considered within alert correlation approaches [139], or relies on formats such as IDMEF to define the unified attributes [134, 66]. In the context of ICS, normalization is rendered difficult by the presence of alerts from both the cyber and the physical domains which are characterized by radically different attributes. Additionally, these alerts from different domains can carry complementary information as in the case of a process-oriented attack originating from the cyber domain. This heterogeneity limits the applicability of existing normalization approaches which cater either for syntactic variations in the representation of an information on the *same attribute*, or the representation of the *same information* in different attributes.

The previous remarks on the heterogeneity and complementarity of the information carried by alerts coming from different domains of an ICS suggest that our main objective is to *enrich* the information gathered from one domain with information from the other domain. In traditional alert correlation approaches [29, 129, 101], **alert enrichment** caters to alerts that suffer from missing information such as temporal information or source IP. To do so, contextual information is added to alerts by relying on knowledge bases [100] which might include information such as the system's topology [129] and

[2] http://tools.ietf.org/html/rfc4765

assets [29]. In the same vein, the approach in [101] uses honeypot databases for contextual information on malware propagation activity or the profile of web servers in order to enrich IDS alerts.

However, because they rely on a unified alert format, these classical enrichment approaches suffer from the same bane as classical normalization approaches. For instance, the approach in [29] fills missing values in attributes with information on the targets, sources, and IDS within the cyber domain. Similarly, the authors in [101] rely on the unified attributes defined by IDMEF to query enrichment information in honeypot databases. The approach in [129] also relies on cyber domain knowledge such as the hop count between hosts and the TCP/IP stack properties of hosts (for instance, the TCP Reset packet acceptance policy). However, such enrichment solutions are possible owing mainly to their confinement to a single domain where attributes are to a great extent homogeneous. For instance, while the approach in [101] aims at enriching local alerts raised by IDS with global threat information available in honeypot databases, all information belongs to the cyber domain and is represented using similar attributes (IP addresses, detection time, classification of attacks) which allows the correlator to query honeypot databases for enrichment information. In contrast, information carried by physical domain alerts is represented in terms of widely different attributes (actuator/sensor events and states) in comparison to information within cyber domain alerts.

B) Verification

This step consists in verifying and discarding alerts which are not pertinent with respect to the system [75]. For instance, alerts might refer to attacks which exploit vulnerabilities that are not applicable to the system. Following the classification in [75], verification approaches can be classified as either active or passive. Active alert verification [75] dynamically looks for information which might determine the pertinence of an alert. On the other hand, passive alert verification [129] relies on a priori information gathered about the system which is possibly stored in formal knowledge bases such as M4D4 [100]. So far, verification approaches geared towards ICS remain rare. As a preliminary step, the development of knowledge bases that might support passive alert verification has been recently studied in the literature [78, 77].

C) Aggregation

Aggregation consists in reducing the number of alerts by grouping alerts together into *meta-alerts*. For instance, alerts which refer to the same attack and come from similar or heterogeneous IDS can be grouped into meta-alerts. The approaches which look for similarity between alerts [63, 139] often rely on a direct comparison between common attributes using different metrics. The aggregation of alerts can serve as a preliminary stage to the identification of attack scenarios through the association of meta-alerts with elementary attack steps. This association can be based on a specification of the expected attacks using for instance correlation rules [99], attack graphs [123] or trees [47]. Attack reconstruction can also be performed by reasoning on pre-conditions and post-conditions

of elementary attack steps [104].

Ultimately, the above approaches assume that alerts are characterized by a subset of common attributes in order to define their similarity metrics. As argued previously, this means that a direct application of these approaches in the context of ICS, where alerts coming from different domains do not share attributes, is difficult without a proper pre-treatment step.

Another issue with classical alert correlation approaches rests in the choice of a *time window* in the case of online aggregation, i.e. where alerts are merged upon reception and immediately forwarded to the next correlation stage. For each new alert, the correlator needs to decide which previously received alerts will be tested for aggregation. A naive approach would memorize and test all received alerts. However, due to resource limits, most alert correlation approaches set a sliding time window with a fixed size. Each new alert is tested with all or a subset of the previous alerts in the current window. For instance, the authors in [140] heuristically set a window size of 2s. More importantly, the choice of a time window determines which alerts will be compared for aggregation. Thus, a time window implicitly defines a similarity measure for alerts based on the time attribute. Alerts which fall within the same time window are assumed to be more similar than those falling outside of the time window. Ultimately, the choice of a window size depends on the goal of the correlation. For instance, the reconstruction of long-term attacks requires a time window with a different scale in comparison to the time window required for the aggregation of alerts corresponding to an attack's elementary steps.

In ICS where the evolution of the physical process is hard to predict, deciding on a single optimal window size is problematic. For instance, the duration of a particular stage in the physical process's execution can depend on the state of other process stages or on the occurrence of a manual intervention by process operators. On the other hand, alerts which occur during the same stage can belong to a single elementary attack step within a more complex multi-stage attack. It is thus important to dynamically adjust the aggregation window depending on the state of the physical process to avoid missing useful aggregations.

D) Impact and priority analysis

In this step, alerts or meta-alerts are evaluated and ranked depending on criteria such as the potential impact of attacks or the confidence in the IDS's verdicts. For instance, the authors in [17] evaluate the priority of alerts in ICS depending on the nature of the targeted components (for example, PLCs are given higher criticity scores), and the ICS zone where the attacks are detected (for example, areas closer to the physical process are deemed more critical).

Discussion All in all, the particular context of ICS imposes several challenges to traditional alert correlation approaches. Chief among these challenges is the need for adequate alert enrichment approaches to associate information within alerts coming from both the cyber and the physical domains. Without such a normalization step, subsequent alert

correlation tasks such as aggregation cannot be carried. Moreover, when performing online aggregation of alerts, the necessity to adjust the aggregation alert window depending on the state of the physical process is essential to avoid missing interesting aggregations. To the best of our knowledge, there are no approaches in the literature which tackle these specific issues within ICS. In Chapter 4, we introduce an approach to perform cross-domain aggregation of heterogeneous alerts in ICS.

2.3.3 Abstraction models

In our cross-domain aggregation approach discussed in Chapter 4, we rely on the notion of *abstraction* to capture the relation between the physical and the cyber domains within an ICS. Our interest in abstraction stems from the observation that, due to the possibilities through which an operator or an attacker can interact with a PLC to affect any given process variable, a violation at the level of the physical domain can correspond to many possible violations at the cyber domain. This disparity between the physical and the cyber domain needs to be taken into account when enriching an alert from one domain with information from the other domain. Otherwise, if the enrichment process does not sufficiently capture all possible cyber domain cases for a given physical domain alert, useful aggregations might be missed in subsequent stages.

To capture this intuition of abstraction relations between domains, many theories of abstraction have been explored in the literature [126, 46, 116]. The majority of these theories, while providing sound approaches to the concept of abstraction, do not delve into the technicalities involved in performing abstraction [126]. As an expedient to this lack of applicability of general theories of abstraction, the $\mathcal{KRA}$ model [126] introduces adaptable procedures, called *abstraction operators*, which correspond to recurring abstraction tasks common to various domains (machine learning, planning, problem solving, etc.).

Within $\mathcal{KRA}$, abstraction is performed on observations containing objects belonging to a domain (also called a representation framework). A domain is represented by the generic types of objects which can be observed, their attributes and their relations. For instance, within the cyber domain, *network flow* can be considered as an object type characterized by attributes such as the *source IP, destination IP,* and *destination port*. In this case, a relation *sameSource* might associate flows that have the same source host. Given a domain, we can generate the space of all possible observations which correspond to all the possible valuations of the objects' attributes and relations within the domain. In our example, this would correspond to all possible network flows within the system and their relations.

Abstraction operators link two domains such as several observations in the first domain, called the *concrete* domain, are linked to a single observation in the second domain, called the *abstract* domain. Each elementary abstraction operator is formally defined in terms of its effect on domains and observations. Once elementary abstraction operators are composed into complex abstraction processes, transforming an observation is performed automatically. For instance, an abstraction operator might *hide* an attribute from the concrete domain. In the network flow example, an abstraction operator might hide the *source IP* attribute. This would generate a new abstract domain where the ob-

served objects are missing the *source IP* attribute. As a result, two observed flows at the concrete domain which differ only in their *source IP* would be mapped to the same object at the abstract domain. This blurring at the abstract domain between observations distinguishable at the concrete domain captures the notion of abstraction within $\mathcal{KRA}$. A formal treatment of this notion of abstraction, along with an inventory of abstraction operators defined over the $\mathcal{KRA}$ model, is given in [126].

In Chapter 4, we take inspiration from the notion of abstraction operator as introduced in the $\mathcal{KRA}$ model and define our own operators. We depart from the $\mathcal{KRA}$ model in that, given our alert enrichment use case explained in Chapter 4, we require the definition of *concretization* operators that map abstract observations to the set of possible concrete observations. On the other hand, the $\mathcal{KRA}$ model is primarily aimed for a bottom-up use; it does not define concretization operators for each abstraction operator.

2.4 Conclusion and positioning

In this chapter, we have presented a taxonomy and a literature overview of intrusion detection approaches for ICS. We have also discussed the issue of correlation and its relevance in ICS. The state of the art on IDS approaches in ICS shows a great preponderance of anomaly-based approaches. Such approaches are interesting because of their ability to detect novel attacks. However, in particular with respect to learning-based approaches, some difficulties subside [130]: (i) the high cost in terms of false positives, (ii) the inadequacy or absence of learning data, (iii) the poor semantic interpretation of the alerts, and (iv) the variability of data on which detection is based. In addition, as exhibited by our exploration of the state of the art, the majority of ICS-based approaches suffer from one or more of the following weaknesses:

- **Detection limited to cyber level observations.** A number of research efforts aim at detecting process-oriented attacks based solely on observations at the cyber level [23, 148, 48]. However, these approaches are not adequate both in terms of attack detection and alert comprehension. For instance, some approaches [23, 48] attempt to model the ordering of network messages between supervisors and PLCs. Any message which violates the ordering captured by the model is considered suspicious and leads to an alert. Given the number of possible messages, simplifications are needed to keep the models tractable. This leads to inaccurate models prone to poor detection results. Low-level events such as network delays can also cause false alerts. Moreover, since the alerts are expressed in terms of cyber attributes (i.e fields in an ICS protocol) rather than physical process attributes (i.e states of actuators and sensors), evaluating the significance of the alerts with respect to the physical process is challenging. As a result, a security operator might struggle to qualify an alert as either true or false. With respect to process-oriented attacks, accurate and understandable detection results require observations at the level of the physical process.

- **Limited model expressivity.** Some approaches [94, 19] which operate directly

at the level of the physical process suffer from insufficiently expressive models that limit the threat model. For instance, the approach developed in [94] can only detect global invariants valid over all the states of the physical process (ex. the property that a temperature should never exceed a maximum threshold anytime during the execution of the process). However, some attacks can violate invariants that are local to specific states of the physical process (ex. a valve which can only be opened during a specific state of the physical process). To detect such attacks and expand the threat model, more expressive formalisms are required.

- **Cost of models.** Many approaches rely heavily on security operators to manually specify the detection models [128, 149]. In addition to the time required to build these models, it is assumed that the operators have adequate knowledge of the formalisms and can specify the models without errors. However, this task becomes increasingly difficult as more complex models are used to cover a larger threat model. To reduce the cost of developing the models, automatic learning approaches need to be explored.

The above weaknesses need to be confronted with the recent trend in ICS tailored attacks. In particular, the analysis of some recent and notable attacks (see Section 1.3) has shown the increasing sophistication in terms of (i) process knowledge (Stuxnet, CrashOverride), and (ii) the move towards extensible attack frameworks supporting multiple protocols (CrashOverride). This recent trend calls for alert correlation approaches that can link attack manifestations from both the physical and the cyber domains.

2.4.1 Positioning

In the remainder of this manuscript, we aim at answering some weaknesses raised above. We argue that process-oriented attacks require suitable intrusion detection approaches that operate directly on actuator and sensor states. Moreover, such an approach needs to use a suitable formalism and relieve the operator from the burden of building the detection models. The development of such an IDS is the main object of Chapter 3.

However, process-domain alerts do not allow the operator to trace the source of the attack. Thus, process-domain alerts need to be correlated with cyber-domain alerts so that the operators can better determine the cause of the process deviations. Existing alert correlation approaches are not adapted for such a task as they cannot deal with the heterogeneity of alerts in ICS. While classical alert correlation models include alert normalization and enrichment as early phases in the correlation process, existing approaches either implicitly assume alerts or rely heavily on the homogeneity of alert attributes which often based on a common format such as IDMEF [3]. However, such homogeneity assumptions, while valid within the confines of the cyber domain distinctive of traditional IT systems, come under question when considering cyber-physical systems such as ICS.

Thus, a major hurdle facing alert correlation in ICS is the pre-treatment of alerts coming from both the cyber and physical domains. This step is crucial to the overall

[3] http://tools.ietf.org/html/rfc4765

goals of alert correlations such as the aggregation of alerts, the reconstruction of scenarios and the reduction of false positives. In Chapter 4, we develop a cross-domain aggregation approach that complements our intrusion detection approach introduced in Chapter 3. Together, both of these approaches can: (i) detect process-aware attacks using process-oriented monitors and, (ii) correlate alerts from the physical and the cyber domains.

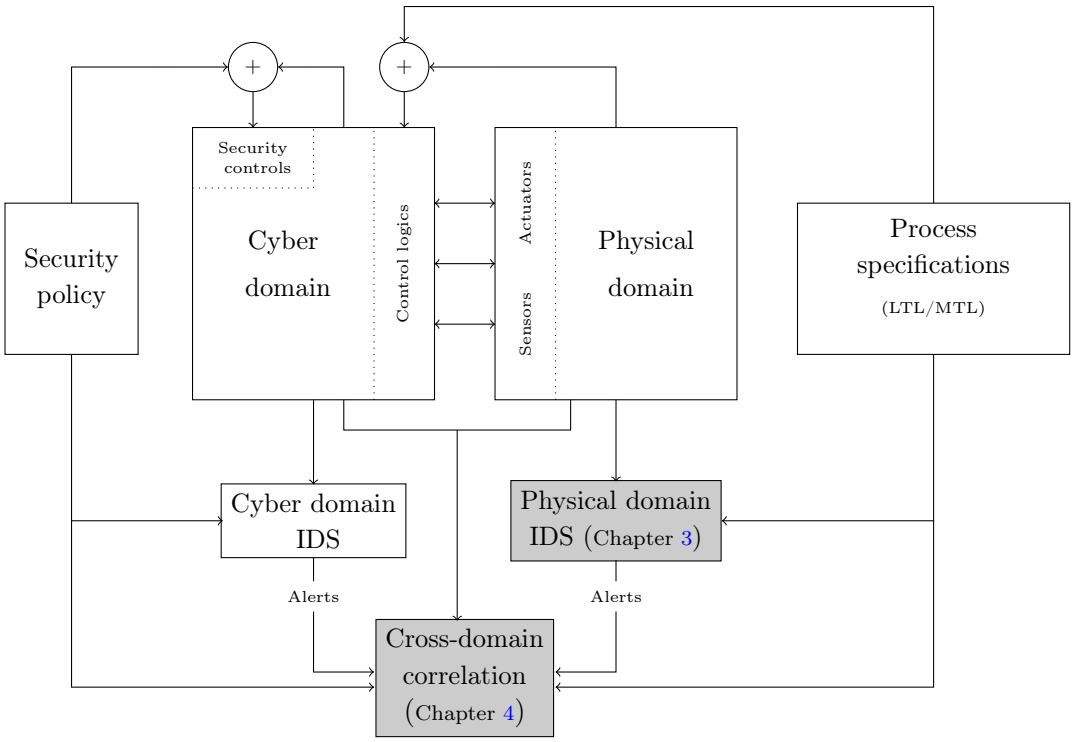


Figure 2.7: Global overview of our contributions

Figure 2.7 gives a global view of the positioning of our contributions with respect to the components of an ICS. In general, an ICS is constituted of both a cyber and a physical domain. In the cyber domain, security controls are deployed to enforce the domain's security policy and to detect, through cyber domain IDS, its violations by an attacker. At the boundary between the cyber and the physical domains, control logics are executed by controllers which manipulate actuators and gather data from sensors in order to steer the physical process. The trajectories given by the evolution of the sensor and actuator states constitute the behavior of the physical process. This behavior, which can be either continuous or sequential, is characterized by temporal constraints known also as process specifications. In this thesis, we focus on the process specifications relative to the sequential behavior as captured by LTL and MTL properties. The way in which these properties are automatically inferred and monitored is the subject of Chapter 3. Finally, given the alerts raised by both the cyber domain and the physical domain IDSs, we perform cross-domain correlation to aggregate manifestations of attacks in both domains.

The way in which alerts are selected, mapped from one domain to another and aggregated is the subject of Chapter 4.

Part II

Contributions

Chapter 3

Process-oriented Intrusion Detection in Sequential Control Systems

Contents

In the preceding chapter, we have provided an overview of common process-oriented intrusion detection approaches found in the literature and identified some of their major weaknesses.

To answer these weaknesses, we introduce in this chapter an anomaly-based intrusion detection approach which : (i) operates directly at the level of the physical process by

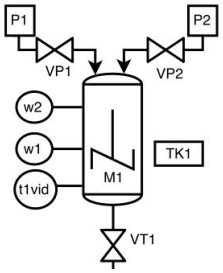


Figure 3.1: Simple sub-process displaying both continuous and sequential behavior

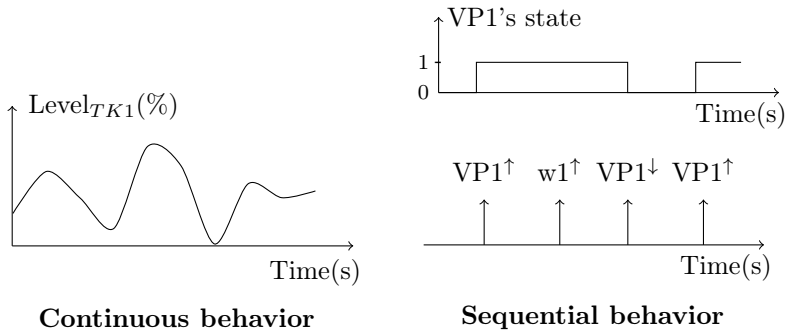


Figure 3.2: Example of continuous and sequential behavior in a physical process

monitoring the state of actuators and sensors, (ii) is based on expressive formalisms that can detect sophisticated attacks covering a broad threat model, and (iii) is fully automatic thus reducing the cost of model specification.

Before exploring in details our approach, we first define our threat model in Section 3.1, in particular, the class of attacks which we aim at detecting. In Section 3.2, we delve into the details of our mining and monitoring approach. We then present our evaluation of the approach in Section 3.3 and finish with some conclusions in Section 3.5.

3.1 Threat model

Industrial control systems are dynamical systems whose state variables, continuous or discrete, vary over time. The behaviors produced by these systems correspond to trajectories in the state space. Informally, a trajectory is a (possibly infinite) sequence of system states and events. Special attention must be given to the hybrid nature of ICS which induces both *continuous* and *sequential* behaviors. The continuous behavior is traditionally modeled through differential equations producing continuous trajectories. On the other hand, the sequential behavior is modeled through discrete event systems producing discrete sequences of logical actuator/sensor states and events.

Example 3.1. *(Continous and sequential behaviors)* Consider the example sub-process in Figure 3.1. This simple system consists of a tank TK1, four actuators (three valves VP1, VP2, VT1, and a motor M1), and three sensors (w1, w2, t1vid). The objective is to synthesize a product in tank TK1 starting from initial products P1 and P2. The synthesis procedure is as follows: first, valve VP1 is opened to introduce a certain quantity of product P1 in TK1. The introduction of a correct quantity of product P1 is signaled by a rising edge of sensor w1. Then, valve VP1 is closed and valve VP2 is opened so that an adequate quantity of product P2 is added to the content of TK1. Afterward, the products in TK1 undergo a mixing phase using motor M1 for a period of 60 seconds. Finally, the resulting product can be collected for further processing through valve VT1.

During the synthesis procedure, the level of tank TK1 varies continuously as products are added and removed. In terms of behavior, the evolution of TK1's level is a continuous trajectory (see Figure 3.2). On the other hand, the control of the process throughout its execution is also based on events (rising edges of sensors) and logical states (open and closed valves). The resulting timed sequence of logical states and events (see Figure 3.2) collectively constitute the sequential behavior of the system.

System engineers are often interested in determining whether the behaviors of a dynamical system are *correct*. Correction is defined as the occurrence or non-occurrence of temporal patterns, also called *specifications* [90]. For instance, a specification can define a correct trajectory as a trajectory where a forbidden state is never reached (i.e overflowing a tank, going over a certain temperature threshold, etc). Based on this notion of correctness, we can define a process-oriented attack as *an attack where the objective is to induce incorrect behaviors at the level of the physical process*. Process-oriented attacks might target both the continuous and sequential aspect of ICS.

Example 3.2. *(Process-oriented attacks)* In the sub-process depicted in Figure 3.1, an attacker can attempt to overflow the tank TK1 or modify the speed of motor M1. These are attacks which target the continuous aspect of the sub-process as they lead to an anomalous deviation in the trajectory of continuous variables (tank level and motor speed). The attacker might also tamper with the sequence of the synthesis procedure in order to generate a product of bad quality. For instance, the attacker might open valve VP2 before valve VP1 is closed to corrupt the balance of products P1 and P2 in the tank. The attacker might also interrupt the mixing phase earlier than expected by stopping motor M1 short of 60s. These attacks target the sequential aspect of the sub-process as they lead to violations in temporal constraints over discrete states and events (for example, the forbidden occurrence of a rising edge of VP2 between a rising edge of VP1 and a rising edge of sensor w1). Stuxnet and CrashOverride (see Section 1.3) are examples of real process-oriented attacks, targeting respectively centrifuges in nuclear plants and circuit breakers in electric grids.

The detection of attacks targeting the continuous aspect has been studied in the literature [55, 1, 132, 22]. These approaches aim at detecting abnormal deviations in the evolution of continuous variables using autoregressive models [55] or through estimated process invariants [1]. In comparison to the continuous aspect, the sequential aspect of

the physical process remains relatively unexplored [83]. Yet, such attacks [122, 76, 83] have been identified as particularly dangerous and difficult to detect using traditional intrusion detection approaches.

Attacks which target the sequential aspect of the physical process are also known in the literature as *sequence attacks* [122, 76] and are sub-divided into two classes: (i) *qualitative* sequence attacks, and (ii) *quantitative* sequence attacks. In a qualitative sequence attack, an attacker manipulates the relative order of commands to produce incorrect behavior. For example, in Figure 3.1, the attacker can issue a particular sequence of commands which result in a wrong balance of products in TK1. In a quantitative sequence attack, the attacker plays on the timing between commands. For example, an attacker might damage a valve by issuing a sequence of commands which repeatedly open and close the valve in a short period of time. The recent CrashOverride attacks [32] comprised both qualitative and quantitative sequence attacks. CrashOverride's qualitative attacks consisted of sending commands to keep circuit breakers open in the hope of inducing outages. CrashOverride's quantitative attacks forced the electric grid's automated protective operations to isolate substations due to the continuous toggle of circuit breakers between open and closed states. Note that in both cases, the individual commands are legitimate (opening a circuit breaker can happen in the absence of an attack), but the temporal ordering of the commands issued by the attacker induces incorrect behaviors.

We focus in this work on detecting process-aware attacks targeting the sequential aspect of the physical process. In the remainder of this chapter, we develop and evaluate an approach to automatically and efficiently infer and monitor temporal constraints pertaining to the sequential aspect of a physical process.

3.2 Mining and monitoring approach

In this section, we present our mining and monitoring approaches to efficiently infer specifications and monitor their violations. We first provide a general overview of our approach while motivating the need to avoid naive inference approaches before delving into more details regarding our mining and monitoring approaches.

3.2.1 Overview

In this section, we present a general overview of our approach which spans two phases : (i) a mining phase where process specifications are inferred from execution traces of the physical process, and (ii) a monitoring phase where the inferred properties are deployed to detect violations.

A) Mining phase

Figure 3.3 shows the general overview of the mining phase. At first, we collect attack-free execution traces that record the evolution of sensors and actuators states throughout

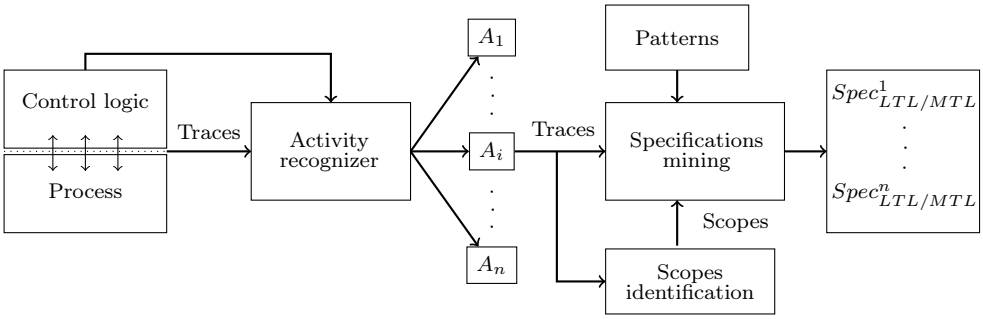


Figure 3.3: General overview of the mining approach

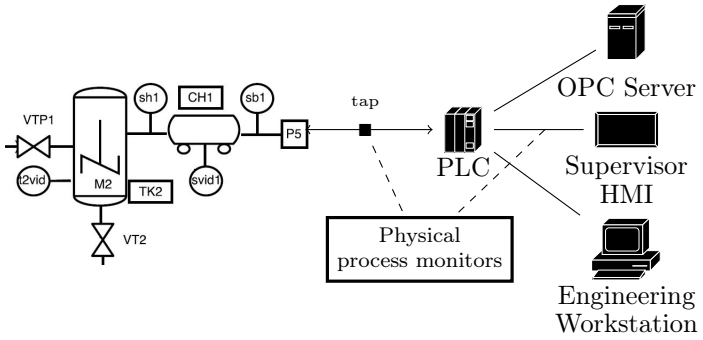


Figure 3.4: Possible placements of physical process monitors within an example ICS

the execution of control logics. Figure 3.4 shows some possible placements of monitors within an example subprocess. Ideally, the state of sensors and actuators can be accessed through the communications between the sensors/actuators and the PLCs using an adequate tap. In terms of security, this would allow the monitors access to the correct state of actuators and sensors even if the PLC is corrupted. On the other hand, setting up a tap might turn out to be impractical if the number of actuators and sensors is consequent, or if introducing a tap would cause an unacceptable disruption in the control loops. If such a tap is not possible for practical purposes, we can also monitor the actuators/sensors signals on each PLC using the supervisory HMI traffic. However, the monitors might not have access to all possible actuator and sensor states depending on which variables are queried by the HMI, and an attacker might blind the monitors if the controllers are corrupted. In both cases, the state of sensors and actuators are periodically sampled and constitute our execution traces.

We focus in our approach on control logics implemented as Sequential Function Charts (SFC) (see [64] for an introduction to SFCs) as they are particularly suitable for sequential processes. We distinguish between the control flow of an SFC, represented by selection and parallel branches, and linear step-transition sequences without branching and parallel execution which we call *activities*. Figure 3.5 provides an example of an SFC and its

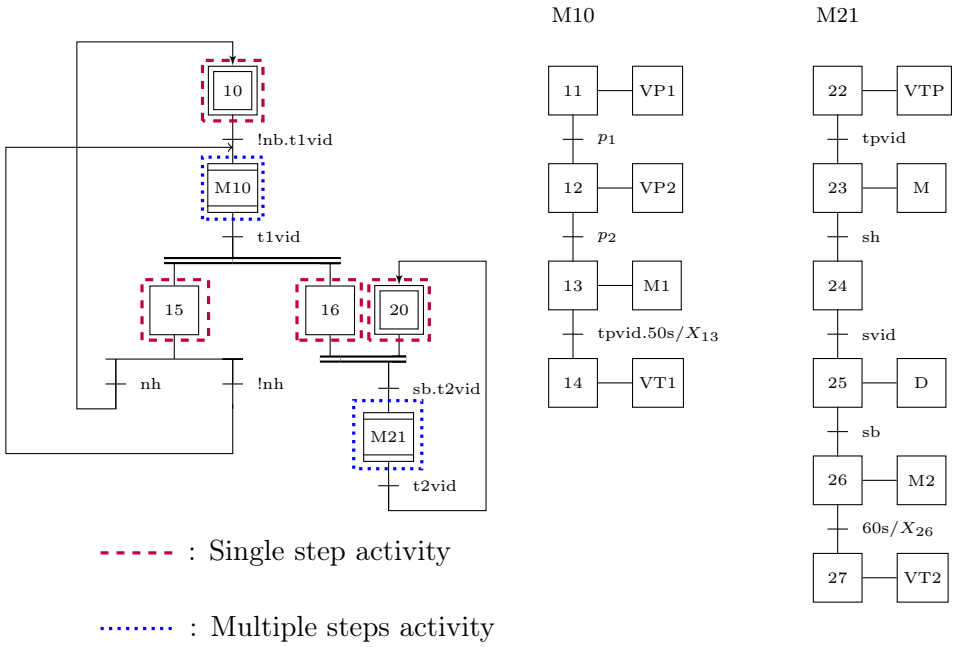


Figure 3.5: Example of an SFC and its constituent activities

decomposition in terms of activities. This SFC contains 7 activities whose size ranges from a single step (steps 10,15,16 and 20) to multiple steps (macro steps M10 and M20). We use the SFC's structure to divide the execution traces per activity (performed by the activity recognizer block in Figure 3.3), and mine process specifications per activity. The activity recognizer reads the execution traces and uses the SFCs to identify the current active steps and thus the current activities. We focus on activities as they represent the actual sequencing of actions performed by the PLCs, while control flow decides which activities are to be executed.

Our goal is to automatically find safety temporal specifications that are valid on attack-free execution traces of the system. This problem is known as *specification mining*. Specification mining approaches [79, 84, 147] typically use specification patterns to find the properties that are valid on the traces. For instance, consider the pattern A *never occurs between* B *and* C. Here, A, B, and C are variables which take values in a set of events that might correspond to changes in the states of actuators and sensors. In particular, B and C define the *scope* of the property, i.e the subsequences of the execution trace where the constraint applies, and A specifies the event which never occurs within the scope. If $B \in \{e_0, e_1, e_2, e_3\}$, $C \in \{e_0, e_1, e_2, e_3\}$, and $A \in \{e_4\}$, then possible instantiations of the pattern include : (i) e_4 *never occurs between* e_0 *and* e_2, (ii) e_4 *never occurs between* e_1 *and* e_3, (iii) e_4 *never occurs between* e_1 *and* e_2. The objective is to find all instantiations that are valid on the attack-free traces.

However, a common issue with specification mining approaches is the significant number of resulting valid instantiations. For instance, the authors in [79] discuss several

techniques to reduce the complexity of the specification mining task. However, the number of valid mined specifications can still remain significant.

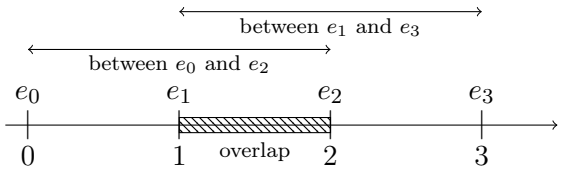


Figure 3.6: Scopes overlap illustration

We argue that many of these instantiations are redundant due to *scope overlaps*. An overlap occurs between two scopes when they delimit intersecting subsequences of the execution trace. We illustrate this situation in the previous example using the execution trace in Figure 3.6. Each position is associated with an event and the figure shows the subsequences corresponding to the scopes *between* e_0 *and* e_2 and *between* e_1 *and* e_3. These scopes overlap since the subsequences they delimit intersect on the subsequence spanning positions 1 and 2.

To illustrate how such redundancies impact the number of violations sent to an operator, suppose that the properties (i), (ii) and (iii) above are valid. If an attack causes event e_4 to happen between position 1 and 2, property (iii) would be violated. However, due to redundancy, properties (i) and (ii) would also be violated. Thus, instead of a single violation, the operator would need to deal with three redundant violations. This problem worsens as the number of events and scopes become large as in the case of long activities. The operator then risks becoming overwhelmed with the significant number of violations. Instead, we propose to mitigate redundancies through a pre-selection of scopes before mining. Only instantiations on the pre-selected scopes are tested for validity. This pre-selection is done by the scopes identification block in Figure 3.3.

Another common issue involved in mining approaches relates to the *falsifiability* of properties. A falsifiable property with respect to a trace is a property which *can* be violated on the trace. Falsifiability is especially relevant with regards to scopes: all properties which refer to non-existent scopes are not falsifiable. Since they specify constraints on non-existent scopes, one cannot check the violation of such properties. Consider for instance the property $universality\_after(valve1, motor1)$. It corresponds to the following LTL formula: $\Box(valve1 \Rightarrow \Box motor1)$. The antecedent of the implication refers to the scope. If $valve1$ is not true at any position on the trace, the implication becomes vacuously true and the formula is not falsifiable. In addition, properties such as $absence\_before(valve1, motor1)$ and $absence\_between(valve1, valve2, motor1)$ will also be vacuously true on the trace since all these formulae involve implications with false antecedents ($\Diamond valve1$ for the first formula and $valve1 \wedge \Diamond valve2$ for the second formula). By performing a pre-selection of scopes, we can avoid mining unfalsifiable properties since all the pre-selected scopes must exist in the execution traces used for mining. This is important since unfalsifiable properties can lead to an increase in the number of mined and subsequently monitored properties. And since they are unfalsifiable, such properties

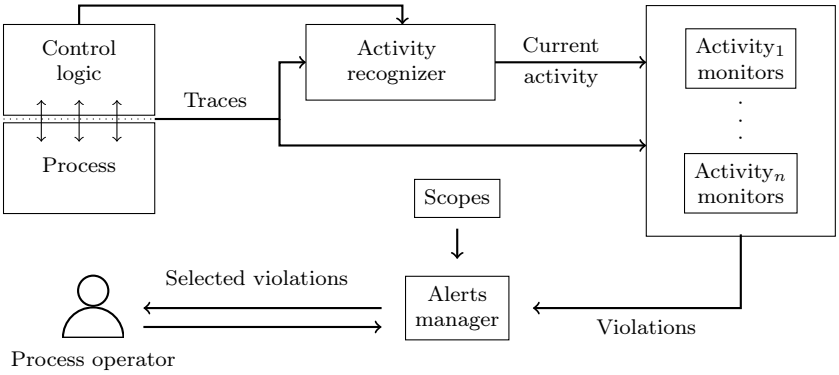


Figure 3.7: Overview of the monitoring approach

cannot be reliably used to represent constraints over the behavior of the system.

B) Monitoring phase

Figure 3.7 depicts the monitoring process. Properties mined in the mining phase are synthesized as monitors (see Section 2.2.3 for a discussion on monitoring techniques) and the execution traces recorded at deployment time are directed to the proper monitors through the activity recognizer. Thus, only the monitors corresponding to the current activities read the execution trace. When a violation is detected, the monitors send a report to an alerts manager which can be a SIEM (Security Information and Event Management) product for instance. The security operator decides whether it corresponds to a false or a true alert given the activity, the scope, and the impacted actuators.

3.2.2 Fundamental definitions

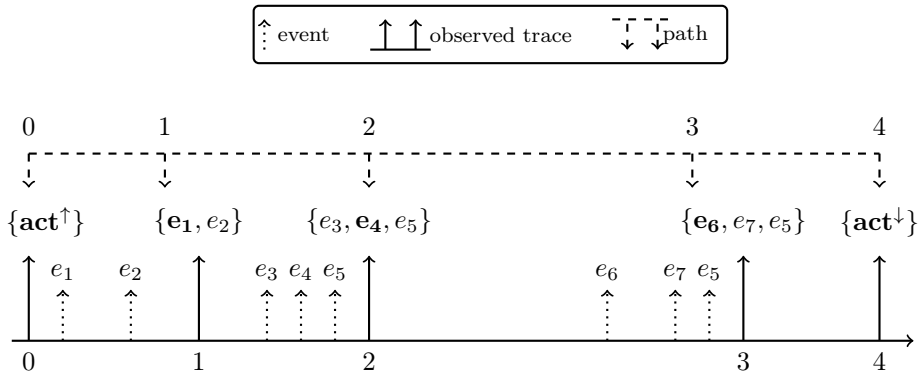


Figure 3.8: Relation between events, observed traces and paths

We now present in details the scopes selection methodology used in the mining phase to mitigate redundancies. To begin, we introduce some fundamental definitions. Let AP be a finite set of boolean variables where each variable refers to a state of a component (sensor or actuator) in the system. Components with continuous values are discretized and mapped to boolean variables. For instance, a level sensor tl reporting continuous values can be discretized to boolean states ($tl \geq th$) and ($tl < th$) with th a threshold. *Events* are changes in the states' values. For a state $a \in AP$, $a^{\uparrow}$ (resp. $a^{\downarrow}$) corresponds to a rising edge (resp. falling edge) of state a.

From the set AP, we generate the set of all possible component events $E = \bigcup_{a \in AP} \{a^{\uparrow}, a^{\downarrow}\}$. We also define the boolean variables $act \in Act$, which indicates whether the activity act is currently active. To mark the beginning (activation of the first step) and the end (deactivation of the last step) of an activity act, we use a pair of events $(act^{\uparrow}, act^{\downarrow})$.

Our view of the process is based on a periodic sampling of actuator and sensor states. Independently of the sampling location, events are always observed at some discrete points in time. An observed trace necessarily begins with the activation of an activity and ends with its deactivation. These activity-specific events do not occur anywhere on the trace. Figure 3.8 provides an example of an observed trace of size 4 over an activity act. Each position in the observed trace corresponds to a set of observed events. For instance, events $\{e_1, e_2\}$ are mapped to position 1 while events $\{e_3, e_4, e_5\}$ are mapped to position 2.

Definition 3.3 (Observed trace)**.** Let $\mathcal{P}(\cdot)$ denote the power set operation. An *observed trace* t of size $n \in \mathbb{N}^*$ over an activity act is an application :

$$t : \{0, 1, \ldots, n-1\} \rightarrow \mathcal{P}(E \cup \{act^{\uparrow}, act^{\downarrow}\})$$

$$\begin{cases} t(0) = \{act^{\uparrow}\}, t(n-1) = \{act^{\downarrow}\} \\ \forall 0 < i < n-1, t(i) \ \cap \{act^{\uparrow}, act^{\downarrow}\} = \emptyset \end{cases}$$

In what follows, we consider t to be an observed trace of size $n \in \mathbb{N}^*$ over an activity act. We refer to the domain $\{0, 1, \ldots, n-1\}$ of t as $Dom(t)$.

Recall that our objective is to reduce the number of mined process specifications by avoiding scope overlaps. To characterize scope overlaps, we formally introduce the notion of scope and scope cover.

Definition 3.4 (Scope)**.** A *scope* is a pair of events (e_1, e_2) where $e_1, e_2 \in Act \cup E$

The cover of a scope returns all the pairs of ordered time positions in the observed trace where the events making up the scope occur. For instance, in Figure 3.8, the scope (e_1, e_6) has cover $C_t((e_1, e_6)) = \{(1, 3)\}$ since e_1 occurs at position 1 and e_4 occurs at position 3. Note that given a scope s, the size of its cover $|C_t(s)|$ can be greater than 1. For instance, in Figure 3.8, $C_t((e_1, e_5)) = \{(1, 2), (1, 3)\}$.

Definition 3.5 (Scope cover)**.** Let $s = (e_1, e_2)$ be a scope. The cover of s over t, denoted by $C_t(s)$, is the set :

$$C_t(s) = \{(i, j) \in Dom(t)^2 | i < j \wedge e_1 \in t(i) \wedge e_2 \in t(j)\}$$

3.2.3 Requirements on the set of scopes

A general approach to specification mining [79] would explore the space of all possible properties given the set of actuators and sensors in the system. However, as discussed in Section 3.2.1, this leads to a significant number of redundancies. Our main objective is to avoid such redundancies through a careful pre-selection of the scopes in S such that they do not exhibit overlaps.

However, this desired non-redundancy property needs to be balanced with other requirements. For instance, we also need to ensure that the scopes in S collectively cover each observed trace and that no property is missed. Otherwise, an attack might go unnoticed either because a subsequence is not covered by any property, or because all possible properties on some subsequence have not been mined. Intuitively, this means that we need to limit redundancies without sacrificing coverage.

In the following, we discuss three requirements that match these observations: (i) maximizing the precision of the scopes, (ii) avoiding unconstrained parts of an observed trace, and (iii) avoiding scope redundancies.

Requirement (i) deals with the issue of missed properties which can occur when the scopes are too broad, i.e their covers span too many time positions within the observed trace. To illustrate this case, consider the set of scopes $S = \{(act^{\uparrow}, e_1),\ (e_1, act^{\downarrow})\}$ on Figure 3.8. Suppose that a property p holds on the pair of time positions $(1, 3)$ but not on $(3, 4)$. Then p does not hold on $(1, 4)$. Since the cover of $(e_1, act^{\downarrow})$ is $\{(1, 4)\}$, mining the properties directly on S would miss the fact that p holds on $(1, 3)$. Note that a similar situation occurs when a scope covers more than one pair of time positions; for instance, if a scope covers two pairs of time positions and the property p holds on one pair but not the other. For example, consider the set of scopes $S = \{(act^{\uparrow}, e_5), (e_5, act^{\downarrow})\}$ on Figure 3.8. The cover of $(act^{\uparrow}, e_5)$ is $\{(0, 2), (0, 3)\}$. If a property p is valid on $(0, 2)$ but not on $(0, 3)$, then p would not be mined for the scope $(act^{\uparrow}, e_5)$ and the fact that p is valid on $(0, 2)$ would be missed. To avoid this issue, the scopes need to be of maximal precision, i.e their cover must span only one pair of time positions whose endpoints are as close as possible.

An effect of this precision requirement is an increase in the number of scopes which ultimately leads to more inferred properties. For instance, if a property p is valid on the 4 intervals in Figure 3.8, then our precision requirement would lead us to infer 4 properties (one for each scope covering a single interval), instead of one global property on the scope $(act^{\uparrow}, act^{\downarrow})$. However, we argue that these additional properties allow the operator to get a more precise information on the location of a violation within the trace. Otherwise, to locate the violation, the operator would need to have access to complete timestamped trace along with a timestamped alert so as to locate exactly the violation within the trace. This puts an additional burden on the operators and requires time synchronization mechanisms between the specification monitors and the trace recorders.

From requirement (ii), the cover of the selected scopes should span all time positions in each observed trace. Otherwise, attacks might be missed if they occur within uncovered time positions. The set of scopes $S = \{(e_1, e_4),\ (e_4, act^{\downarrow})\}$ does not cover all time positions in the observed trace of Figure 3.8 since the combined covers of (e_1, e_4) and

$(e_4, act^{\downarrow})$ is $\{(1,2),(2,4)\}$ which does not include $(0,1)$.

Redundancies (requirement (iii)) happen when two scopes overlap, i.e when the intersection of their covers is not empty. For instance, in Figure 3.8, the scope (e_1, e_4) with cover $\{(1,2)\}$ overlaps with the scope (e_1, e_6) with cover $\{(1,3)\}$ since their cover intersect on $(1,2)$. Avoiding redundancies is our main objective since it leads to a reduction in the number of deployed monitors which otherwise overload the operator with redundant information.

We now give a formal expression of the three requirements through the following two propositions.

Proposition 3.6 (Requirement (i))**.** A set of scopes S is of maximal precision over trace t iff
(a) $\forall s \in S, |C_t(s)| = 1$.
(b) For every other set of scopes S' satisfying (a) :

$$\forall (e'_i, e'_j) \in S', \exists (e_i, e_j) \in S \mid$$

$$\begin{cases} C_t((e_i, e_j)) = \{(c_i, c_j)\}, C_t((e'_i, e'_j)) = \{(c'_i, c'_j)\} \\ c'_i \leq c_i < c_j \leq c'_j \end{cases}$$

Condition (a) in Proposition 3.6 restricts the cover of a scope to a single pair of time positions. Condition (b) in Proposition 3.6 says that for any scope (e'_i, e'_j) in a set of scopes S', S contains a scope (e_i, e_j) at least as precise as (e'_i, e'_j). For instance, consider the two sets of scopes $S = \{(act^{\uparrow}, e_4), (e_4, act^{\downarrow})\}$ and $S' = \{(act^{\uparrow}, act^{\downarrow})\}$ on Figure 3.8. We have $C_t((act^{\uparrow}, act^{\downarrow})) = \{(0,4)\}$, $C_t((act^{\uparrow}, e_4)) = \{(0,2)\}$, and $C_t((e_4, act^{\downarrow})) = \{(2,4)\}$. Since the time position pairs $(0,2)$ and $(2,4)$ are within $(0,4)$, the scopes in S are more precise than S' on $(0,4)$.

Proposition 3.7 (Requirements (ii) and (iii))**.** Let $I = \{(i,j) \in Dom(t)^2, j - i = 1\}$ be the set of all successive time position pairs in $Dom(t)$. A set of scopes S satisfying Proposition 3.6 is non-redundantly covering a trace t iff [1] :

$$\forall (i,j) \in I, \exists! s \in S \mid C_t(s) = \{(k,l)\} \wedge k \leq i < j \leq l$$

Proposition 3.7 makes sure that each pair of time positions on t is covered once and only once by a scope in S. For instance, regarding Figure 3.8, the set of scopes $S = \{(act^{\uparrow}, e_4), (e_1, act^{\downarrow})\}$ does not satisfy Proposition 3.7 since $(1,2)$ is covered by two scopes : $(act^{\uparrow}, e_4)$ and $(e_1, act^{\downarrow})$.

3.2.4 Generation of a set of scopes for a single observed trace

In Section 3.2.3, we discussed the requirements which a set of scopes needs to meet. In this section, we show how such a set of scopes can be generated first for a single observed

[1] $\exists!$ means *there is one and only one*

trace. We postpone the discussion on the generation of a set of scopes for sets of observed traces to Section 3.2.5.

To generate a set of scopes satisfying the requirements in Section 3.2.3 for a single observed trace, we introduce the notion of a *path*. A path is an application which uniquely associates each time position in the observed trace with a single event. This event must not appear anywhere else in the observed trace. The goal of a path is to uniquely characterize, using an event, every time position in the observed trace. Using these characterizing events, we will show that a set of scopes satisfying Propositions 3.6 and 3.7 exists.

In Figure 3.8, a possible path derived from the observed trace associates $act^{\uparrow}$ to position 0, e_1 to position 1, e_4 to position 2, e_6 to position 3, and $act^{\downarrow}$ to position 4. Given this path, e_1 uniquely identifies position 1, i.e e_1 is never observed at any other position in the observed trace. Events e_4 and e_6 perform a similar task for positions 1 and 2. On the other hand, position 2 cannot be associated with event e_5 since e_5 occurs also in position 3.

Definition 3.8 (Path)**.** A *path* p_t over t is an application :

$$p_t : Dom(t) \rightarrow E \text{ such that :}$$

$$\begin{cases} \forall i \in Dom(t), p_t(i) \in t(i) \\ \forall (i,j) \in Dom(t)^2, i \neq j \implies p_t(i) \notin t(j) \end{cases}$$

Given an observed trace t, we denote by Π_t the set of all possible paths over t. Note that given an observed trace t, the number of paths $|\Pi_t|$ can be greater than 1. In the previous example, position 1 could have been associated either with e_1 or e_2. Thus, different paths are compatible with the observed trace in Figure 3.8. In what follows, we refer to the image of a path p_t over t as $Img(p_t)$.

Remark 3.9 (Path bijection)**.** From Definition 3.8, p_t defines a bijective application from $Dom(t)$ to $Img(p_t)$ since each time position in $Dom(t)$ is associated with a unique event in $Img(p_t)$.

Remark 3.10 (Existence of a path over an observed trace)**.** Generally, one cannot guarantee the existence of a path over an observed trace. A path cannot be generated if one of the time positions can only be assigned events appearing elsewhere on the observed trace. In this case, the position cannot be uniquely characterized. Later, we discuss a solution to this issue which identifies and isolates a minimal subset of events that hinders the generation of a path. In the worst case, our solution reduces to a standard specification mining approach which explores all possible scopes. However, we will show that significant improvements can still be achieved due to the regularity of the observed traces in control systems.

Proposition 3.11. Given an observed trace t a set of scopes S satisfying Propositions 3.6, and 3.7 exists if a derived path exists.

Discussion. Let t be an observed trace and $p_t \in \mathbb{P}_t$ a derived path. By setting $S = \{(p_t(0), p_t(1)), (p_t(1), p_t(2)), \cdots, (p_t(n-2), p_t(n-1))\}$, we can show that S satisfies Propositions 3.6 and 3.7.

- Condition (a) in Proposition 3.6 is satisfied due to the bijectivity of p_t which maps every scope in S to a single pair of time positions.
- Regarding condition (b) in Proposition 3.6, let S' be another set of scopes satisfying condition (a) in Proposition 3.6. Let $(e'_1, e'_2) \in S'$ and (c'_1, c'_2) the unique pair of time positions (by condition (a)) to which $C_t((e'_1, e'_2))$ reduces. Since, by construction, S covers all successive time positions in $Dom(t)$, we can always map (e'_1, e'_2) to a scope $(e_1, e_2) \in S$ with $C_t((e_1, e_2)) = \{(c_1, c_2)\}$ such that $c'_1 \leq c_1 < c_2 \leq c'_2$.
- Finally, concerning Proposition 3.7, let $(i, j) \in I$ be a pair of successive time positions in $Dom(t)$. By construction, the scopes in S cover all time positions in $Dom(t)$. Thus, there exists a scope in S, namely $(p_t(i), p_t(j))$, covering (i, j). This scope is unique due to the bijectivity of p_t.

Algorithm 1 shows our procedure for generating the set of possible paths $\mathbb{P}_t$ of an observed trace t. The procedure incrementally generates the paths for t by constructing intermediate paths for t's prefixes. Lines 2-5 first initialize $\mathbb{P}_t$ with a separate sequence for each of the events in the first position of t. Lines 6-15 extend the sequences in $\mathbb{P}_t$ with an additional event for each of the remaining positions in t. The variable $newPaths$ holds, after each iteration of the main for-loop in Line 6, the paths constructed for the prefixes of length i of t. When paths for a prefix cannot be constructed, i.e. $newPaths$ is empty after an iteration of the main for-loop, then no path can be generated for t and the procedure returns an empty result.

3.2.5 Generation of a set of scopes for a set of observed traces

Up to this point, we have only considered how to generate a set of scopes S satisfying the requirements in Section 3.2.3 for a single observed trace. However, we generally have several observed traces each corresponding to a run of an activity. Moreover, these observed traces might not be all identical, i.e they might contain different events or ordering of events. Our objective in this section is to examine how to generate a set of scopes when we have several distinct observed traces of an activity.

To generate S for a set of observed traces $T = \{t_1, t_2, \cdots, t_n\}$, we rely on the corresponding set of possible derived path sets $\Pi_T = \{\mathbb{P}_{t_1}, \mathbb{P}_{t_2}, \cdots, \mathbb{P}_{t_n}\}$ as shown in Figure 3.9. Here, each $\mathbb{P}_{t_i}$ is generated from the corresponding observed trace t_i as discussed in Section 3.2.4. Then, the scopes identification step takes as input the observed paths in Π_T and determines a set of mining scopes. We focus, in the remainder of this section, on the scopes identification step.

One straightforward case occurs when there exists a common path for $t_1, t_2, \cdots, t_n$, i.e $\bigcap_{1 \leq i \leq n} \mathbb{P}_{t_i} \neq \emptyset$. Here, since there exists a common path to all observed traces, the set

Algorithm 1: Path generation procedure (*generate_path*)

Data: t : Observed trace
Result: $\mathbb{P}_t$: Paths generated from t

```
1  ℙ_t ← {};
2  for j ∈ t[0] do
3      σ ← sequence();
4      σ[0] ← j;
5      ℙ_t.add(σ);
6  for 1 ≤ i ≤ |p| − 1 do
7      newPaths ← {};
8      for j ∈ p[i] do
9          for σ ∈ ℙ_t do
10             if j ∉ Range(σ) then
11                 σ[i] ← j;
12                 newPaths.add(σ);
13     if |newPaths| == 0 then
14         return None;
15     ℙ_t ← newPaths;
16 return ℙ_t;
```

of scopes can be generated as in the case of a single observed trace which was discussed in Section 3.2.4.

A more challenging situation arises when a common path cannot be found. To illustrate this situation, Figure 3.10 shows an example of four observed traces for a single activity where no possible paths can be derived. While the first and third observed traces contain 6 observation points, the second and fourth observed traces only 7 observation points. Moreover, between observed traces of the same size, no common paths can be generated. For instance, position 2 on the first and fourth observed traces cannot be characterized by the same event.

Upon closer inspection, we note that while events e_1, e_2, e_3, e_4, and e_5 occur, *relative to one another*, at the same positions across all the observed traces, event e_6 is less stable. In fact, e_6 does not occur at all on the first observed trace and occurs at the third or second position in the remaining traces. This means that scopes with event e_6: (i) do not occur at every observed trace and, (ii) do not cover the same time positions across all the observed traces. For instance, the scope (e_6, e_3) does not occur on the first observed trace and covers different time positions on the remaining observed traces. Thus, such scopes cannot be used to construct a set of scopes satisfying the requirements in Section 3.2.3.

We call events such as e_6 *conflicting events*. Our solution consists in identifying a minimal set of conflicting events whose removal allows for the generation of a common path. The resulting path produces the largest possible set of scopes which still satisfies

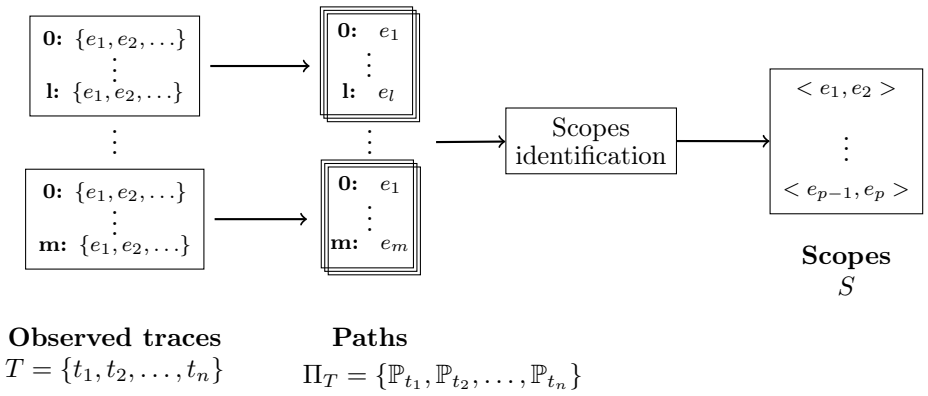


Figure 3.9: Scope selection approach based on paths derived from the observed traces

the requirements in Section 3.2.3. The discarded conflicting events are used with a standard mining approach with no guarantee of non-redundancy, i.e all possible scopes including the conflicting events are explored. The main drawback of this solution is the increase in the number of redundant alerts. In the worst case, if all events are conflicting, our solution reduces to a standard mining approach with redundancies.

Due to the regularity of the observed traces generated from activities, we expect the set of conflicting events to be minimal. As we discuss in our evaluation analysis (see Table 3.4 in Section 3.3), the practicality of this approach rests on two observations regarding the regularity of the observed traces : (i) the proportion of unique observed traces is small with regards to the total number of recorded observed traces (less than 6 % on average) which limits the size of the set of observed traces from which a common path needs to be found, and (ii) the proportion of conflicting events is limited, especially in the case of redundancy-prone long activities, which leads to significant gains in terms of avoiding redundant properties. While these observations have only been made with respect to our evaluation testbed, we expect similar observations on datasets coming from operational plants. Thus, in practice, substantial improvements can still be expected in terms of redundancy compared to a standard approach.

Algorithm 2 shows our procedure for resolving conflicting events. The algorithm attempts to find a common path by ignoring one or more events in the originally observed traces. It takes as input the set T of observed traces and the set of events E (see Section 3.2.2). Then, it attempts to find the lowest number of events which must be ignored in the originally observed traces to find a common path. In Line 1, the algorithm allocates a queue to hold the current candidate events to be ignored. The queue is initialized in Lines 2-3 with each event in E, i.e the smallest possible sets of conflicting events. In Line 4, the algorithm uses a set *visited* to keep track of candidate sets that have already been explored. Then, while there are still candidate sets (Line 5), a *current* candidate is selected from the front of the queue (Line 6) and inserted in the visited set (Line 7). Line 8 computes the projection of each trace in T over the events in *current*, and

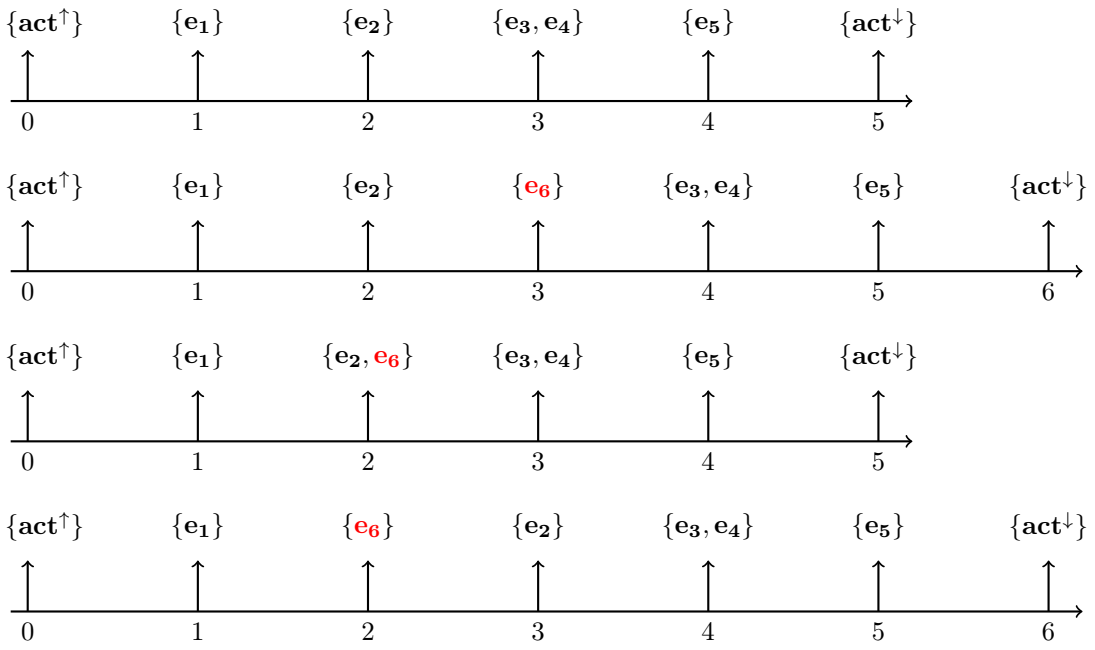


Figure 3.10: Occurrence of a conflicting event on four observed traces of an activity

Line 9 computes the set of paths for each projected trace (procedure *generate_paths* in Algorithm 1). If a common path is found among the paths of the projected traces (Line 10), a solution is found and the algorithm returns it (Lines 11-12). Otherwise, the algorithm adds, if possible, new sets of candidate conflicting events to the queue (Lines 13-15). This is performed by extending the *current* set with one new event in E. If *current* already holds all events in E, or if an extended set of candidate events has already been visited, the set is not added to the queue.

3.2.6 Mining specifications

Recall that our goal is to find the safety temporal properties that are valid on attack-free execution traces of the system, i.e the properties which are not violated by the attack-free execution traces. To avoid the redundancy and falsifiability issues exhibited by naïve mining approaches, we only try to find the safety temporal properties specified over the scopes in the set of scopes S generated by the procedures in Sections 3.2.4 and 3.2.5. The task of mining specifications depends on whether the properties we are looking for are qualitative or quantitative. In this section, we examine the mining task for each case.

Qualitative properties. To mine qualitative properties, we instantiate each pattern by replacing variables with process variable events and states, synthesize each instantiation as a monitor, and run it on the set of training execution traces. A qualitative property is valid if it is *never* falsified at any point in the training traces. Table 3.1 shows

Algorithm 2: Conflicting events resolution

Data: T : Finite set of observed traces
Data: E : Finite set of system events
Result: $\hat{p}$: Common path

$q \leftarrow$ queue();
for $e \in E$ **do**
  $enqueue(q, \{e\})$;
$visited \leftarrow \{\}$;
while q *not empty* **do**
  $Current \leftarrow dequeue(q)$;
  $insert(visited, Current)$;
  $T_{proj} \leftarrow \{project(t, Current), t \in T\}$;
  $\Pi_{T_{proj}} \leftarrow \{generate\_paths(t), t \in T_{proj}\}$;
  $\hat{p} \leftarrow \bigcap_{\mathbb{P} \in \Pi_{T_{proj}}} \mathbb{P}$;
  if $\hat{p} \neq \emptyset$ **then**
    return $\hat{p}$;
  for $e \in E \setminus Current$ **do**
    if $|Current| < |E| \wedge Current \bigcup \{e\} \notin visited \wedge Current \bigcup \{e\} \notin q$ **then**
      $enqueue(q, Current \bigcup \{e\})$;
return $None$;

the qualitative properties we mine, along with the scopes we use and the monitorability of the property. Note that we only mine using the **between** since we look for constraints that are valid between each pair of events defining a scope. Moreover, since we mine properties per activity (between the start of the activity denoted by $act^{\uparrow}$ and its end denoted by $act^{\downarrow}$), the **after** and **after until** are in fact limited by the event $act^{\downarrow}$, the **before** scope is limited by the event $act^{\uparrow}$, while the *global* scope is equivalent to the pair of events $(act^{\uparrow}, act^{\downarrow}$. Thus, for our approach, all the scopes reduce to the between scope.

Quantitative properties. Instantiating quantitative properties is more complicated in comparison to qualitative properties since they involve a temporal parameter. For instance, the property pattern $durationMin(X, T)$ expresses the constraint that the logical state X remains true for at least T units of time. While X can be replaced with an actuator state as in the case of qualitative properties, finding a suitable substitution for T requires more attention. We observe that if the property $durationMin(S, 50s)$ is true for a particular process state S, then $duration(S, 40s)$ is also true. In general, any value of $T \leq 50s$ will be true. Similarly, if $durationMin(S, 70s)$ is false, then $durationMin(S, 80s)$ is also false. Here also, any value for $T \geq 70s$ will also be false. We thus observe that for properties such as $durationMin(X, T)$, there exists a value $\hat{t}$

Table 3.1: Qualitative properties used in our mining approach

Qualitative property	**Scope**	**Monitorability**
Universality	Between	Yes
Absence	Between	Yes
Existence	Between	Yes
Response	Between	Yes
Precedence	Between	Yes

Table 3.2: Quantitative properties used in our mining approach

Quantitative property	**Scope**	**Monotonicity**	**Monitorability**
Maximum duration	Between	Yes	Yes
Minimum duration	Between	Yes	Yes
Bounded recurrence	Between	Yes	Yes
Bounded response	Between	Yes	Yes
Bounded invariance	Between	Yes	Yes

of T such that $durationMin(X, t)$ is true for all values $t \leq \hat{t}$ and false for all values $t > \hat{t}$. We say that $durationMin(X, T)$ is monotonic in the parameter T. Properties which exhibit monotonicity allow for an efficient procedure to find a suitable value for T through a binary search approach. Checking whether a property exhibits monotonicity has been studied in the literature [4, 62]. For instance, the approach in [4] allows identifying monotonic properties based on syntactic deductive rules while the approach in [62] encodes a property as a set of constraints in a fragment of first-order logic and then uses a satisfiability modulo theories (SMT) approach to check for monotonicity. Table 3.2 shows the quantitative properties we mine, the scopes we use, the monitorability of the properties and their monotonicity. For the same reasons as for the qualitative case, we only mine using the *between* scope.

Based on the discussion above, the mining procedure consists in finding, for a user-specified range of values $[t_{min}, t_{max}]$, a user-specified *tightness* level d, and a monotonic property p, a substitution of T which lies within d time units from the true target value $\hat{t}$. The tightness level determines how far from the true target level $\hat{t}$ the final solution can lie. Lower values of d mean that the substitution needs to lie closer to $\hat{t}$. However, values of d that are too low can lead to properties that are too conservative and thus to false positives. By default, if no tighter initial bounds are given by the user, we set t_{min} to the value 0 while we set t_{max} to a value equal to the total duration of the training execution traces. To avoid overfitting the parameter d, we look for a value of d which minimizes the number of false positives on the attack-free validation dataset which is independent of the attack-free training dataset used to mine the properties.

First, the validity of substitutions $p_{t_{min}}$ and $p_{t_{max}}$ over the training execution traces is checked. If $p_{t_{min}}$ and $p_{t_{max}}$ return the same validity verdict (i.e, both invalid or valid), then no suitable substitution for parameter T can be found. Otherwise, we partition the range of values in two and check for the validity of substitutions : $p_{(\frac{t_{min}+t_{max}}{2}+d)}$ and $p_{(\frac{t_{min}+t_{max}}{2}-d)}$ If both substitutions return the same validity verdicts, the search continues on either $[t_{min}, \frac{t_{min}+t_{max}}{2}]$ or $[\frac{t_{min}+t_{max}}{2}, t_{max}]$. Otherwise, a substitution for T within the tightness level d has been found and the search can stop.

3.3 Evaluation

We evaluate our approach on a hardware-in-the-loop testbed with a simulated physical process controlled by real PLCs. The process is a simplified chemical plant producing benzene by hydrodealkylation of toluene [137]. A thorough evaluation would require real data from an operational plant. However, getting such data is difficult due to the particularly sensitive context of ICS. Also, publicly releasing information describing the setup and the dataset is problematic, which is not satisfying from a scientific point of view. Publicly available datasets[2] are often too simple, including few sensors/actuators. Studies which use real datasets are often limited to network trace files, while we require the availability of control logic for a comprehensive analysis. Our goal was thus to develop a realistic dataset which can still be publicly shared so that our results can be reproduced.

The physical process we have developed is simulated using OpenModelica[3]. Its parameters (tank dimensions, heating temperatures, mixing time, etc.) are set so that the physical process undergoes several cycles during our simulations. Control is distributed using three PLCs (Schneider M340 and M580 and Wago IPC-C6 with additional RTU 750-873). Each PLC sends commands and receives sensor information from the real-time OpenModelica simulation via input/output (I/O) interface cards. The control logics executed by each PLC are implemented in SFC. To carry its task, a PLC may need to communicate with other PLCs to query sensor states or send commands for distributed control. Finally, HMI associated with each allows the operators to monitor and perform manual interventions. Communication among all the ICS components is performed using Modbus/TCP.

We now detail the simulated physical process, discuss how we carry attacks and describe the implementation and the datasets. All the datasets, control logics and process simulation files will be available online[4].

3.3.1 Testbed description

The simulated physical process is shown in Figure 3.11. This process takes input products P1 to P5 (on the horizontal line at the top of the figure) and yields output products P6 to P10 (on the vertical line at the right of the figure). The main objective of the process

[2] https://sites.google.com/a/uah.edu/tommy-morris-uah/ics-data-sets
[3] https://openmodelica.org
[4] https://persyval-platform.univ-grenoble-alpes.fr/0/searchbyrecently

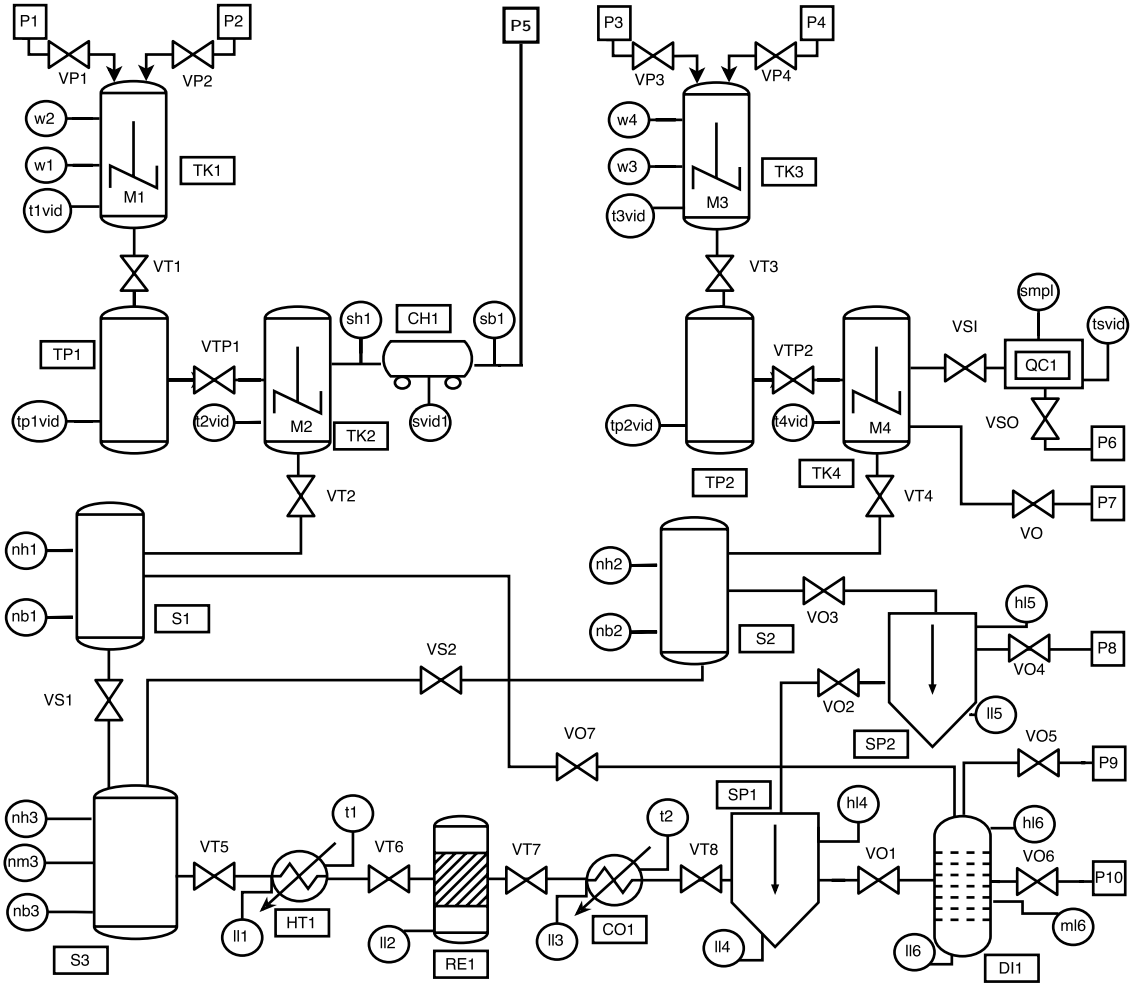


Figure 3.11: Physical process model used in our evaluation

consists in carrying a chemical reaction to synthesize product P10 from reactants in the silos S1 and S2. These reactants are manufactured from initial products in several stages involving mixing (motors M1 to M4) and quality testing (QC1). The reaction occurs in the reactor RE1 and residual reactants are recycled by feeding them back to the silos S1 and S2. Products P1 and P2 undergo a first phase of mixing in tank TK1 using motor M1. The resulting product goes through further treatment in tank TK2 using motor M2 and a secondary product P5 delivered by cart wagon CH1. As soon as the product leaves TK1 for the buffer tank TP1, a new batch of product is prepared until the silo S1 is full. Input products P3 and P4 undergo a similar treatment in tanks TK3, TP2, and TK4. However, before leaving for the silo S2, the mixture is tested for conformity by delivering a sample to QC1. If the product is of good quality, the mixture is transferred

to S2 through valve VT4. Otherwise, the mixture is taken out of the physical process through valve VO (Product P7). The result of the conformity test is indicated by the sensor SMPL and the sample product is evacuated using valve VSO (Product P6).

To carry the chemical reaction, the required amounts of the reactants are taken from the silos S1 and S2 and heated in HT1. The actual reaction occurs in the reactor RE1. Then, the resulting substance is cooled in CO1. Not all the reactants undergo a transformation, so separation and recycling are performed in the separators SP1, SP2 and in the distillation column DI1. The reactants are recycled by feeding them back to the silos S1 and S2. Products P8 and P9 are considered as by-products of the chemical reaction.

The physical process involves 71 sensors and actuators and has two modes: manual and automatic. The manual mode allows the operators to carry interventions on the process and override the control performed by the PLCs. All in all, the physical process is divided into 18 activities: 10 activities for the generation of the reactants in S1 and S2, and 8 activities for the actual chemical reaction and recycling phases.

3.3.2 Attacks and operators interventions

To evaluate our approach, we need to generate attack traffic. However, attacks depend strongly on the monitored system and, contrary to traditional IT domains, no dataset with significant attacks targeting ICS is available. We thus identify possible attacks on sequential control systems, taking into account real attack cases [122, 32]. In particular, we model our attacks on the recent CrashOverride malware [32] by defining sequences of commands whose execution induces incorrect process behavior (i.e, sequence attacks).

Moreover, we need to subject the ICS to interventions from process operators. Including manual operations from process operators is essential as they constitute a major cause of false positives [23]. While false positives are hard to avoid in anomaly-based approaches, our aim is to give security operators enough information so that they can identify whether an alert is actually a false positive. We discuss in this section a model of manual interventions covering both attacks and process operator interventions.

A) General intervention models

We start from the observation that both a process operator's and an attacker's interventions consist of a series of commands sent to a PLC to perform actions following a procedure. For instance, the process operator has access to a high-level view of the physical process and its state through an HMI and uses its interface to affect the process. Yet, on the wire, these high-level operations are seen as a series of commands. In our case, these commands correspond to Modbus messages sent to a PLC.

There are two parameters to consider in such a series : (i) the actual order of the commands, and (ii) the delay between the commands. For a legitimate operator, uncertainty can arise for both of these parameters. From an attacker's perspective, sequence attacks are carried out by manipulating either the order of the commands (qualitative sequence attacks), or the delay between commands (quantitative sequence attacks). To

model the behavior of operators and attackers, we use semi-Markov models which have proven to be successful in modeling operators' behaviors in human supervisory control settings [16]. We refer to these models as *action automata*. Each state in the model corresponds to a step in the procedure followed by an operator or an attacker. Moreover, each state is associated with a Modbus/TCP command to be sent to a PLC. For each state, we specify the parameters (μ,σ) of the normal distribution associated with the state duration. As discussed in [16], the parameters can help distinguish between different supervisory tasks.

Additionally, both operators and attackers may need to carry their actions in specific states of the physical process. For instance, an operator might open a valve to empty a tank only when the tank is full. Similarly, an attacker might wait for a chemical reaction to get underway before manipulating actuators to interfere with it. We allow legitimate actions and attacks to be performed at specific states of the physical process by associating the action automata with steps within the control logics (SFCs) executed by the PLCs. At runtime, when control reaches an SFC step which is associated with an action automaton, a choice is made as to whether to execute the action automaton or not. If the decision is made to execute the action automaton, the commands specified by the automaton's steps are issued. We now examine in more details the action automata corresponding to legitimate operations and attacks.

B) Action automata for legitimate operations

To model legitimate operations, we use a loop-free linear sequence of commands with a random delay (type I action automaton in Figure 3.12). The length of these sequences depends on the number of actuators manipulated by the operator or attacker. In all cases, two states are allocated for setting the process into manual mode at the beginning of the sequence, and back to automatic mode at the end of the sequence. Process operators can either perform quick actions (for instance setting the process into manual mode and moving to the next action in the procedure) or actions with a predefined amount of time (open a valve for 20s). To handle such cases, we adjust the state duration parameters in the operators' automata accordingly. In these examples, we would use the following settings respectively ($\mu = 2s, \sigma = 500ms$) and ($\mu = 20s, \sigma = 2s$)

C) Action automata for attacks

Some attacks can also be modeled using type I automata. In this case, the attacker imitates the behavior of a legitimate operator. In this case, the main differences compared to a legitimate operation reside either in the set of manipulated actuators, the ordering of manipulations, or the SFC steps which are associated with the automaton. Thus, in our model, an automaton corresponding to a legitimate operation can represent an attack if it is associated with the wrong SFC step, i.e if it is executed in the wrong state of the physical process. For instance, manipulating valve *VP1* in Figure 3.11 makes sense when tank *TK1* is being filled but represents a malicious behavior when *TK1* is being emptied.

In addition to the first model, we use a second type of automata (type II action automaton in Figure 3.12) to model a *wear attack* where the attacker rapidly and successively sends two commands such as the opening and closing of a valve or a circuit breaker [32]. At the heart of this model is a single loop oscillating between two states. As in the first type, two states are allocated for moving into and out of manual mode.

Table 3.3 presents examples of attacks that we carry in our evaluation. Annex B provides the details of the parametrization of these attacks. We start by identifying incorrect behaviors which the physical process must not exhibit. This includes, for instance, tampering with motors or injecting a bad product mix. Then, we identify, for each scenario, the actuators which the attacker need to manipulate and the automaton type that models the particular sequence of actions.

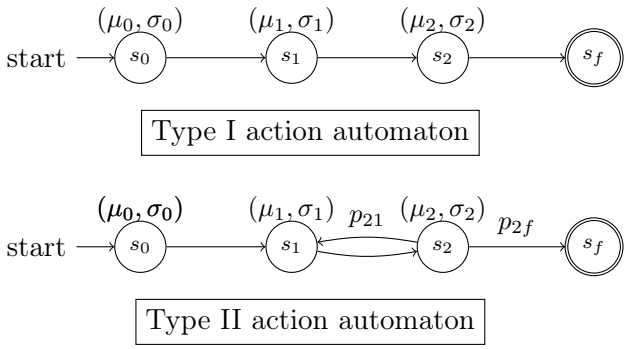


Figure 3.12: Action automata used in the intervention block

3.3.3 Implementation and datasets

Mining, monitoring, and activity recognition are implemented in C++/Python. Analysis is performed on an Intel Dual Core i7 2.6 Ghz machine with 16 GB of RAM and Linux kernel 4.4.0. The monitors are generated using Aerial[5]. The activity recognizer takes two files as inputs : (i) an XML file following the PLCOpen[6] standard containing the specification of the SFC which execution is followed, and (ii) a configuration file which specifies the steps to activities mapping. The algorithms presented in Section 3.2 for the generation of scopes are implemented in Python.

The evaluation relies on 4 network captures during which the physical process goes through several cycles for every activity (up to tens of cycles for the most frequently executed activities). A 2-hours long training traffic is used to generate the set of scopes and perform the mining of properties. Concurrently, we perform a pre-defined set of legitimate operator interventions as discussed in Section 3.3.2. The training dataset reflects realistic conditions where certain legitimate behaviors are absent due to the

[5] https://bitbucket.org/traytel/aerial
[6] http://www.plcopen.org

Table 3.3: Description of attacks carried out in the evaluation

Attack	**Objective**	**Targeted variables**	**Type**
A1	Inserting a bad balance of products P1 and P2 in TK1	VP2@PLC1	I
A2	Inserting a bad balance of products P1 and P2 in TK1	VP1@PLC1 VP2@PLC1	I
A3	Wearing motor M1 through quick start/stop commands	M1@PLC1	II
A4	Introducing unfinished reactant in S1	VT2@PLC1	I
A5	Stopping mixing of reactant in TK2 short of 60s	M2@PLC1	I
A6	Inserting a bad balance of products P3 and P4 in TK2	VP4@PLC2	I
A7	Wearing motor M3 through quick start/stop commands	M3@PLC2	II
A8	Stopping mixing of reactant in TK4 short of 60s	M4@PLC2	I
A9	Introducing bad quality product in S2	VT4@PLC2	I
A10	Tampering with the reactor during the chemical reaction	R1On@PLC3	II

limited training window. For evaluation purposes, we use 3 captures spanning 3 hours each and containing a total of 36 instances of sequence attacks.

3.4 Analysis

In this section, we evaluate our approach along three aspects : (i) the practicality of scopes generation, (ii) the reduction in the number of mined redundant properties, (iii) the detection of attacks and sources of false positives

3.4.1 Scopes generation

Table 3.4: Results of metrics M_1^A and M_2^A for the activities with conflicting events

Activity	M_1^A	M_2^A
Act 1-1	6.7	6.25
Act 1-2	5.1	5
Act 2-1	3.2	6.25
Act 2-2	4.5	8.3
Act 3-1	10	2.9

The first step of the mining process consists of identifying the scopes to be used in the search for valid properties. As we argued in Section 3.2.3, our procedure for generating an adequate set of scopes depends on regularity assumptions about the observed traces. We assess this regularity through two activity-specific metrics : (i) the proportion of unique observed traces with respect to the total number of observed traces

$$M_1^A = \frac{\#\ unique\ observed\ traces\ in\ activity\ A}{\#\ observed\ traces\ in\ activity\ A}$$

and (ii) the proportion of conflicting events

$$M_2^A = \frac{\#\ conflicting\ events\ in\ activity\ A}{\#\ events\ in\ activity\ A}$$

M_1^A gives an indication of the number of unique observed traces over which scopes identification is performed for activity A (see Section 3.2.5). $M_1^A = 1$ means that all the observed traces for activity A are unique, while lower values of M_1^A indicate that the activity tends to generate fewer unique observed traces. As we discussed in Section 3.2.5, the case when multiple unique observed traces are present for a given activity is more complex and requires finding a common path across all the unique observed traces.

M_2^A provides an indication of the number of events which must be discarded to construct a set of scopes for activity A. As M_2^A gets lower, fewer events need to be

discarded which in turns leads to more precision and fewer redundancies in the mined properties.

Table 3.4 reports, for each activity A containing conflicting events, the values of M_1^A and M_2^A. We see that, for activities with conflicting events, the percentage of unique observed traces does not go beyond 10% of the activity's total observed traces.

This indicates that observed traces tend to be stable across the majority of the activities. The only exceptions pertain to activities Act1-2 and Act2-2. These activities correspond to the generation of, respectively, the first and second reactants in tanks TK1 and TK2. This generation depends in turn on the state of buffer tanks TP1 and TP2 (via sensors tpvid1 and tpvid2) which ultimately depends on the amount of reactants in silos S1 and S2. These dependencies which lie outside of the activities Act1-2 and Act2-2 mean that events pertaining to sensors tpvid1 and tpvid2 can happen anywhere in their observed traces. In fact, we observe that, for a given activity, conflicting events generally pertain to sensors whose state does not depend exclusively on actions performed within the activity. However, we also observe that the proportion of conflicting events shrinks as the size of the activity (i.e, number of steps) grows. For instance, in Table 3.4, Act3-1 which is the longest activity within the system (11 steps), exhibits the fewest number of conflicting events (5.5%). Since longer activities tend to produce significantly more redundant properties in naive mining approaches, this means that significant gains in terms of redundancies can be achieved.

Table 3.5: Scopes generation result

Activity	**# SFC Steps**	**Time (s)**	**# Scopes**
Act 1-1	4	0.064	6
Act 1-2	6	0.092	6
Act 2-1	4	0.084	6
Act 2-2	3	0.076	3
Act 2-3	3	0.069	3
Act 2-4	2	0.069	2
Act 3-1	11	1.508	17
Act 3-2	2	0.047	2
Act 3-3	1	0.060	1
Act 3-4	1	0.036	1
Act 3-5	1	0.040	1
Act 3-6	1	0.061	1
Act 3-7	1	0.035	1
Act 3-8	1	0.077	1
Act 3-9	1	0.063	1

Table 3.5 shows the results of the scopes' identification step for the 15 evaluated activities. The table displays the number of SFC steps in each activity, the time elapsed to identify the scopes and the number of identified scopes to be used in the mining phase. We observe from Table 3.5 that the number of mining scopes is always greater

or equal than the number of SFC steps for each activity. We also note that the number of scopes increases with the number of steps within the activity. This indicates that, in general, more precise constraints can be mined on the traces at the level of events and scopes compared to the level of SFC steps. Since the scopes are able to represent finer distinctions within the observed traces, they are more suitable to avoid over-specification issues (see Section 3.2.3).

3.4.2 Properties mining

The second step of the mining process is to find, for each activity, the set of valid properties on the training dataset using the scopes generated in the previous step.

Table 3.6: Interpretation of the set of scopes generated for activity Act 1-1 and examples of mined properties

Scope	**Interpretation**	**Examples of properties**
$(act^{\uparrow}, t1vid^{\downarrow})$	A new batch of the first reactant is being produced, tank TK1 is being filled with product P1	absence(vp2) durationMin(vp1,13s)
$(t1vid^{\downarrow}, p1^{\uparrow})$	Finished filling TK1 with product P1	universality(vp1) durationMax(vp1,44s)
$(p1^{\uparrow}, p2^{\uparrow})$	Filling TK1 with product P2	absence(m1) existence(vp2)
$(p2^{\uparrow}, p2^{\downarrow})$	Tank TK1 has been filled with products P1 and P2, now mixing products in TK1 and started emptying tank TK1	response(m1,vt1) durationMin(m1,50s)
$(p2^{\downarrow}, p1^{\downarrow})$	Tank TK1 is being emptied	universality(vt1) durationMax(vt1,46s)
$(p1^{\downarrow}, act^{\downarrow})$	Tank TK1 is being emptied and the batch of the first reactant has been produced	absence(vp1) durationMin(vt1,28s)

Table 3.6 shows the set of scopes identified for activity Act 1-1 along with the interpretation of each scope and some properties mined on the scope. For instance, the scope $(act^{\uparrow}, t1vid^{\downarrow})$ corresponds to the start of a new batch of the first reactant and the introduction, in tank TK1, of product P1. On this scope, we mine qualitative properties such as $absence(vp2)$ which forbids valve vp2 from being opened while product P1 is inserted (otherwise the balance between products P1 and P2 is impacted). We also mine quantitative properties such as $durationMin(vp1, 13s)$ which specifies the minimum amount of time valve vp1 is opened when introducing product P1.

Table 3.7 reports the mining results for all 15 activities. For each activity, the table displays the amount of time needed for the mining process along with the number of qualitative and quantitative mined properties for the scope-based approach. We also compare

the number of mined qualitative properties with Texada[7], which is a general purpose LTL specification miner which does not perform any filtering of redundant properties. Compared to a general purpose miner, the scope-based approach consistently mines a fewer number of properties. This is especially apparent for long activities such as Act 3-1 where we only retain 0.6% of the properties mined by Texada. Generally, longer activities involve more actuators and sensors, increasing the number of possible scopes and of properties. In our case, for each activity, the number of scopes on which mining is performed is significantly lower than the number of possible scopes. This provides an advantage over other approaches which do not perform any pre-selection of scopes and produce a large number of redundant properties.

Table 3.7: Properties mining result

Activity	**Qualitative properties**			**Quantitative properties**	
	Mining time (s)	Scope-based	LPB15 [79]	Mining time (s)	Scope-based
Act 1-1	5	31	1616	16	19
Act 1-2	7	38	4247	19	18
Act 2-1	6	28	1437	25	16
Act 2-2	4.5	14	216	14	2
Act 2-3	4.5	14	409	11	6
Act 2-4	5	13	381	16	5
Act 3-1	32	281	46657	54	81
Act 3-2	5.5	12	105	8.5	1
Act 3-3	5	12	38	11.5	3
Act 3-4	6	12	38	11	3
Act 3-5	5.5	12	38	8	3
Act 3-6	6	12	38	11	3
Act 3-7	5.5	12	38	8.5	3
Act 3-8	6	12	38	11	3
Act 3-9	5	12	38	11	3

3.4.3 Attack detection and false positives

To evaluate the performance of our intrusion detection approach, we deploy the monitors synthesized from the properties identified in the mining phase and run the three evaluation datasets. As expected, all attacks were successfully identified across all datasets. Table 3.8 reports the properties violated by each of the attacks described in Table 3.3. For each violated property, the operator has access to the scope on which the property is valid and which has a particular meaning with respect to the process (for example see Table 3.6). For instance, attack A3 which consists in a wear attack on motor M1 is detected through a quantitative property on the scope $(p2^{\uparrow}, p2^{\downarrow})$ restricting the minimum amount of time motor M1 must be operating (50s). Except for attacks A4 and

[7] http://bitbucket.org/bestchai/texada

A9, all attacks are detected through a single property violation. However, attack A4 and A9 differ in that attack A4 generates similar violations on consecutive scopes, while A9 reports different violations on the same scope.

Table 3.8: Temporal properties violated for each attack carried in the evaluation

Attack	**Type**	**Violated properties**	
		Qualitative	**Quantitative**
A1	I	($t1vid^{\downarrow}$,$p1^{\uparrow}$):absence(vp2)	
A2	I	($p1^{\uparrow}$,$p2^{\uparrow}$):absence(vp2)	
A3	II		($p2^{\uparrow}$,$p2^{\downarrow}$):durationMin(m1,50s)
A4	I	($tp1vid^{\uparrow}$,$sb^{\downarrow}$):absence(vt2) ($sb^{\downarrow}$,$sh^{\uparrow}$):absence(vt2)	
A5	I		($svid^{\downarrow}$,$act^{\downarrow}$):durationMin(m2,60s)
A6	I	($t2vid^{\downarrow}$,$p3^{\uparrow}$):absence(vp4)	
A7	II		($p4^{\uparrow}$,$p4^{\downarrow}$):durationMin(m3,50s)
A8	I		($tsvid^{\downarrow}$,$act^{\downarrow}$):durationMin(m4,60s)
A9	I	($tsvid^{\uparrow}$,$act^{\downarrow}$):absence(vt4) ($tsvid^{\uparrow}$,$act^{\downarrow}$):universality(vo)	($tsvid^{\uparrow}$,$act^{\downarrow}$):durationMin(vo,30s)
A10	II		($ll1^{\downarrow}$,$ll3^{\uparrow}$):durationMin(r1on,400s)

We also record false alarms due to operator interventions not seen in the training dataset. Generally, we observe one violation per operator action. This follows from the precision requirement on the scope set which ensures that violations report the most precise scopes. Maintaining a limited number of alerts is important to avoid overwhelming the security analyst. In contrast, due to a large amount of redundant mined properties, the other approaches produce 10 to 30 times as much number of violated properties for every operator's action during the longer activities. In this case, the analyst needs to sift through a larger number of violations to discriminate attacks from false alarms. While most activities have a false positive rate below 10%, the longest activities such as Act 3-1, can reach a rate as high as 35%, of which more than 80% are false positives relative to quantitative properties. These quantitative false positives all relate to slight deviations from the quantitative constraint such as when a valve which must not be kept for more than $21s$, is kept for $22s$. One reason for these discrepancies is the limited representativity of the total behavior of the activity afforded by the training traces. By gathering more data on the normal behavior of the activity, we expect such false positives to diminish. We discuss below another solution which consists in using more robust monitoring techniques in the case of quantitative properties.

Consider the following property $durationMin(vp1, 13s)$. A violation of this property can occur if valve $vp1$ stays open for an amount of time less than 13s. However, in terms of interpreting the violation, one can further distinguish between cases where $vp1$ stays open for an amount of time close to 13s (for instance 12s or 11s), and cases where the behavior of the valve deviates significantly from the constraint (for instance by staying open for 3s). While the first case might point to a false positive, the second case can

be more readily interpreted as a wear attack on valve $vp1$. Thus, in addition to the binary information concerning the violation of the quantitative property, an operator might also want to get a quantitative estimation of the deviation with respect to the constraint. Recent research within runtime verification has begun exploring such issues by developing such monitoring techniques for certain classes of metric temporal logic properties [31].

3.5 Conclusion

Detecting process-aware attacks in ICS requires adequate intrusion detection measures which take into account the physical process. In this chapter, we have presented an approach to efficiently mine and monitor safety properties on execution traces of the physical process. Process operators can use the approach both for online detection of attacks, but also for retrospective root cause analysis of abnormal events.

However, relying solely on process-oriented intrusion detection approaches does not allow the operator to identify the source of the attack. Such a global view requires the combination of intrusion detection approaches from both the physical and the cyber domains. Overall, the integration of feedback from the operator on reported violations and of alerts from other IDS might help achieve lower rates of false positives and facilitate the diagnosis of abnormal events. However, correlating alerts from heterogeneous domains as can be found in ICS faces many challenges. In particular, proper pre-treatment and normalization of heterogeneous alerts are necessary to unify the alert attributes and enable alert aggregation and attack scenario reconstruction. In the next chapter, we introduce an alert normalization approach that can assist in correlating alerts from multiple IDS in an ICS.

Chapter 4

Cross-domain Alert Correlation in ICS

Contents

Chapter 3 introduced a physical domain intrusion detection approach which reports alerts in terms of violated temporal constraints over actuator/sensor states and events. However, physical domain detection methods cannot identify the source of the attacks, hence the need for cyber domain IDS which analyze network activity at a higher level of the ICS. On the other hand, these cyber domain IDS do not provide information about the effect of attacks on the physical process. Since attacks targeting the physical process can produce manifestations both in the physical and the cyber domains, we need to correlate manifestations from both domains to better understand the alerts.

In IT systems, alert correlation [139, 29, 140] is a set of techniques used to eliminate redundant alerts, reduce the number of false alerts, and reconstruct attack scenarios. So far, there have been few attempts at developing alert correlation in ICS. Classical alert correlation models [140] include *normalization* and *pre-processing* stages which unify the

attributes' values between alerts from different IDS. However, these stages are often either implicitly assumed, or their extent is limited to filling missing values such as the time, the source or the type of attacks [140]. In ICS, these pre-processing stages are more complex due to the heterogeneity of the alerts received by the correlator from both the physical and cyber domains.

Moreover, for each new alert, a correlator needs to decide which previously received alerts will be tested for correlation. This choice is called an alert selection policy. A naive policy will memorize and test all received alerts. However, due to resource limits, most alert correlation approaches set a sliding time window with a fixed size. Each new alert is tested with all or a subset of the previous alerts in the current window. For instance, the authors in [140] heuristically set a window size of 2s. In ICS where the evolution of the physical process is hard to predict, deciding on a single optimal window size is problematic.

In this chapter, we develop a correlation solution which bridges between the physical domain intrusion detection approaches such as the one developed in Chapter 3, and cyber domain intrusion detection approaches. To pre-process alerts in ICS, we use information about the devices laying at the boundary of the physical and cyber domains, namely the controllers. Given the diversity of the network interfaces and ICS protocols supported by the controllers, an attacker can use different vectors at the network level to achieve the same effect on the physical process. Based on these observations, we argue that the physical and cyber domains are linked through abstraction relations. By relying on the notion of abstraction operator [126], alerts from the physical domain can be rewritten in terms of cyber attributes and correlated with cyber domain alerts. Next, to answer the issue of alert selection, we develop alert selection policies that adapt to the state of the physical process rather than use a fixed size time-based alert window. Finally, through an aggregation phase, we group alerts from both the physical and the cyber domains into clusters that reflect elementary attack steps or operator interventions. Our solution can then serve as the necessary first stage for more other alert correlation tasks such as attack scenario reconstruction.

This chapter is organized as follows. Section 4.1 presents our alert correlation solution. Section 4.2 describes our evaluation setup and implementation. Section 4.3 provides a discussion and analysis of the results. Finally, Section 4.4 concludes the chapter.

4.1 Alert correlation approach

In this section, we delve into the details of our alert correlation solution. First, we pinpoint the scope of our approach before going over our contribution.

4.1.1 Scope of our alert correlation approach

Figure 4.1 shows the scope of our alert correlation process. For simplicity, we omit the verification and alert prioritization stages which can also be integrated within the correlation process.

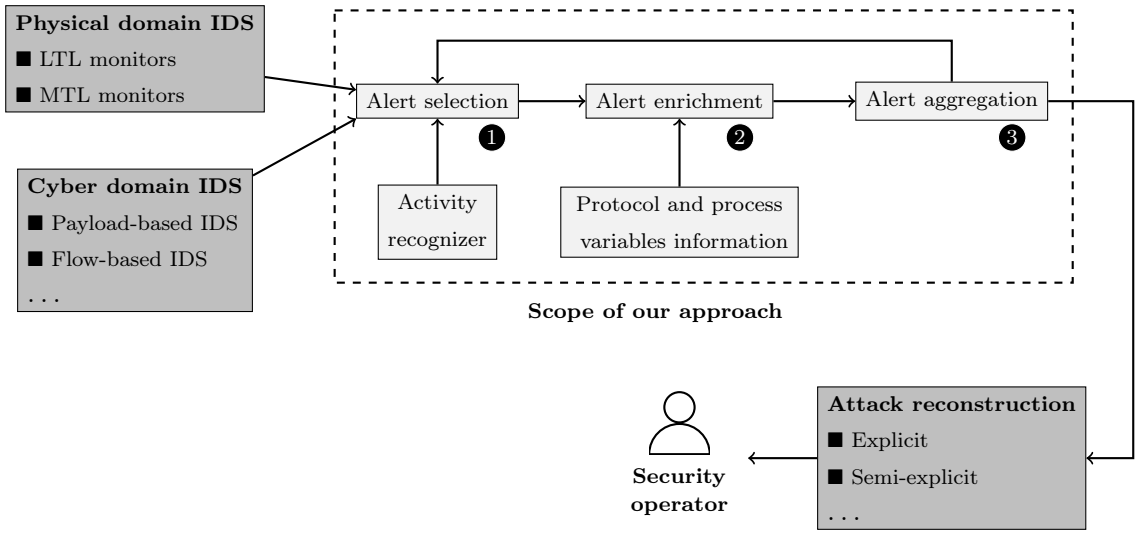


Figure 4.1: Scope of our alert correlation approach

The input of our approach consists of alerts coming from both the physical domain and the cyber domain IDS. Besides physical domain IDS such as the one developed in Chapter 3, various cyber domain IDS, many of which were discussed in Chapter 2 (flow-based IDS, vocabulary and grammar oriented IDS, etc.), can be used as alert sources for the correlation block.

The output of our approach is a set of meta-alerts, each one aggregating several alerts which eventually come from different domains. The meta-alerts can be used in subsequent correlation stages to reconstruct complex attack scenarios that can be displayed to a security operator through a SIEM.

The heart of the alert correlation process is performed in three stages. In the *alert selection* stage (❶), the received alerts are inserted into activity-specific alert windows (see Section 3.2.1 for a discussion of activities). Since we consider an online correlation process, alert windows limit the number of alerts considered for normalization and aggregation. In our approach, these alert windows are instantiated and terminated depending on the evolution of the activities at runtime. The *activity recognizer* is responsible for notifying the correlator about the runtime context of the physical process (i.e, the active steps within the activities). It tracks the current active steps using the SFCs implemented in the PLCs and the evolution of actuator/sensor states.

The activity recognizer is provided with the SFCs executed by the PLCs and does not need to connect to the PLCs in order to obtain them. Similarly, it can directly monitor the evolution of actuator and sensor signals without interacting with the PLCs. Thus, through the activity recognizer, the alert selection stage has access to the runtime context of the physical process independently of the PLCs.

In the *alert enrichment* stage (❷), the correlator transforms the alerts coming from different IDS so that they can be compared for aggregation. In particular, alerts from

the physical domain are transformed and enriched with cyber domain information given knowledge about the PLCs' configurations and supported protocols.

Finally, the *alert aggregation* stage (❸) tests whether alerts are correlated and keeps track of the correlations. Correlated alerts are grouped into meta-alerts which can be presented to the security operator through a SIEM software for instance. The meta-alerts contain references to the aggregated alerts. Feedback about successful aggregations is reported back to the *alert selection policy*. This feedback can be used to select the next alerts to test for aggregation (edge from ❸ to ❶).

4.1.2 Motivating example

Figure 4.2 depicts a simplified sub-process that will be used in this section as a running example to illustrate our alert correlation approach. This sub-process represents a single processing stage in a multi-stage chemical plant. Product incoming from the physical process is first put in tank *TK2* through valve *VTP1*. Using the cart *CH1*, a quantity of product *P5* is also added to the content of tank *TK2*. Then, the products are mixed using motor *M2* for a fixed amount of time. Finally, tank *TK2* is emptied using valve *VT2*.

These control operations are performed by a PLC and correspond, in terms of control logics, to a single activity (a linear sequence of step-transitions). In the supervisory domain, a supervisor HMI along with an OPC server allows the operators to perform manual interventions on the process. In our example, the supervisor HMI can communicate with the PLC using either Modbus or SOAP web-services through the PLC's internal web server. The OPC server only uses Modbus. The engineering workstation is solely used to reprogram the PLC and is never used for any process interventions.

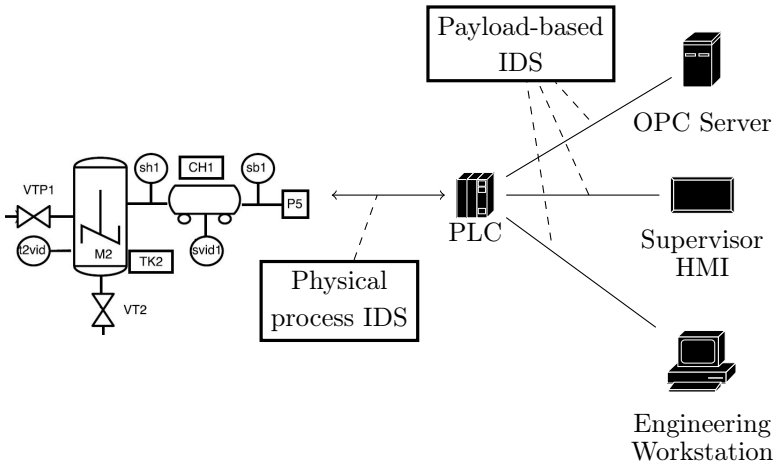


Figure 4.2: Example of a subprocess with its associated control and supervisory devices

Two types of IDS are deployed : (i) a physical process IDS which surveys the state

of the sensors/actuators through cyclical observation of the signals and reports any process specification violation such as the opening of a valve in the wrong step, and (ii) a protocol-specific payload-based IDS which reports any unknown attempt to manipulate an actuator on the PLC. For instance, unknown attempts might be due to a Modbus write command sent from an unauthorized host (engineering workstation), or by using a previously unseen protocol (a SOAP request from the OPC Server).

A possible process-aware attack on this sub-process is an attempt to overflow the tank *TK2* through the malicious manipulation of valve *VTP1*. In a typical run, this valve is only manipulated at the beginning of the activity until tank *TK2* is filled. To overflow *TK2*, an attacker can keep valve *VTP1* open throughout the activity. If the attacker sends the command to open valve *VTP1* from an unauthorized host such as the engineering workstation, then both the physical process IDS and the payload-based IDS will raise alerts. The goal of alert correlation is to group such alerts so that they can be displayed together to a security operator. The alert from the payload-based IDS would allow the operator to incriminate the engineering workstation and further investigate the source of the attack (by examining the workstation's logs for instance). However, the payload-based IDS alert can also be a false positive (a message not observed during learning). Similarly, the opening of valve *VTP1* might be a legitimate action or an action with low incidence on the physical process. Using the alert raised by the physical process IDS, the operator can recognize that it is indeed an attack and that given the current context of the physical process, opening valve *VTP1* is critical. Thus, the association of information from both the payload-based IDS and the physical process IDS allows the operator to gain precious time in the characterization of the alerts.

In the remainder of this section, we introduce an alert correlation approach that maps the physical domain alerts into the cyber domain so that it can be compared and associated with the other cyber domain alerts.

4.1.3 Alert selection

The alert selection phase relies on an alert window and some predefined alert selection policies to decide which alerts will be provided to the subsequent stages.

Since we perform online correlation, i.e. where alerts are correlated upon reception and the results are immediately forwarded to the next stage, each alert can only be kept a finite amount of time during which it can be aggregated with other alerts. Performing correlation online answers two pressing constraints: (i) the limits in terms of memory resources, and (ii) the need to reduce the response time of the correlator. Since alerts are stored in volatile memory, only a limited number of alerts can be kept at any time by the correlator. Additionally, reducing the response time of the correlator allows the security operators to react in a timely fashion.

To answer these constraints, classical alert correlation [140] approaches resort to use a sliding time window. Only alerts that fall within this window are retained by the correlator. While such a solution answers the above requirements, agreeing on the size of the time window is not straightforward. Ultimately, the choice of a window size depends on the goal of the correlation. For instance, the reconstruction of long-term

attacks requires a time window with a different size in comparison to the time window required for the aggregation of alerts corresponding to an attack's elementary steps. For a given correlation goal, the choice of the time window size determines which alerts will be compared for aggregation. Thus, a time window implicitly defines a similarity measure for alerts based on the time attribute. Alerts which fall within the same time window are assumed to be more similar than those falling outside of the time window. Selecting an inadequate time window size can lead to missing correlations if the size is too short, or to unwanted correlations if it is too long.

Because the evolution of the physical process is hard to predict, deciding on a single optimal time parameter is problematic. For instance, the duration of a particular stage in the physical process's execution can depend on the state of other process stages or on the occurrence of a manual intervention by process operators. As a result, interesting correlations can be missed if the windows are not adjusted accordingly. For example, in the sub-process shown in Figure 4.2, maintaining the valve *VTP1* open causes violations to be raised by the physical process IDS throughout the activity since *VTP1* should only be manipulated at the beginning of this activity, i.e. when filling the tank. A sliding time window-based correlator would need to adjust the window's size to match the activity's duration in order to collect all the alerts relative to the attack. However, the transitions in an activity can depend on other parts of the physical process (the influx of product *P5* from another subprocess) as well as on specific time conditions (such as running motor *M2* for a certain amount of time). Moreover, an operator can also manipulate *VT2* to interrupt the flow of output product and thus impact the activity's duration. Consequently, the time spent in each activity is hard to predict.

We thus argue that a time-based alert window is not adequate in the context of ICS. Instead, we propose to dynamically adjust the size of the alert window depending on the state of the physical process to avoid missing useful aggregations. As argued above, the selection of an alert window policy depends on the overall goal of the correlation. In our case, we focus on the early stages of the correlation process where the goal is to aggregate alerts corresponding to the elementary steps of complex long-term attacks (see Section 4.1.1). In the context of ICS, long-term attacks include distributed attacks that can impact multiple activities or stages within the physical process. Regardless of the complexity of a given attack, its elementary steps always target an actuator in a particular activity. Our goal is to correlate alerts corresponding to these elementary steps. To do so, instead of setting a fixed alert window size, we associate an alert window to each activity and adjust the size of the window at runtime by monitoring the activation of steps in the activity. As a result, the size of a given alert window varies dynamically depending on the time taken by its associated activity.

To illustrate our solution, Figure 4.3 shows, for the activity corresponding to the subprocess in Figure 4.2, the activity's linear sequence of steps and transitions along with its associated time window. The steps, labeled *S0*-*S4*, are associated with actions on actuators in the physical process. For instance, at step *S0*, the PLC opens valve *VTP1* to let input product in tank *TK2*. The transitions between steps depend on boolean conditions over sensor states and internal variables. For example, when control

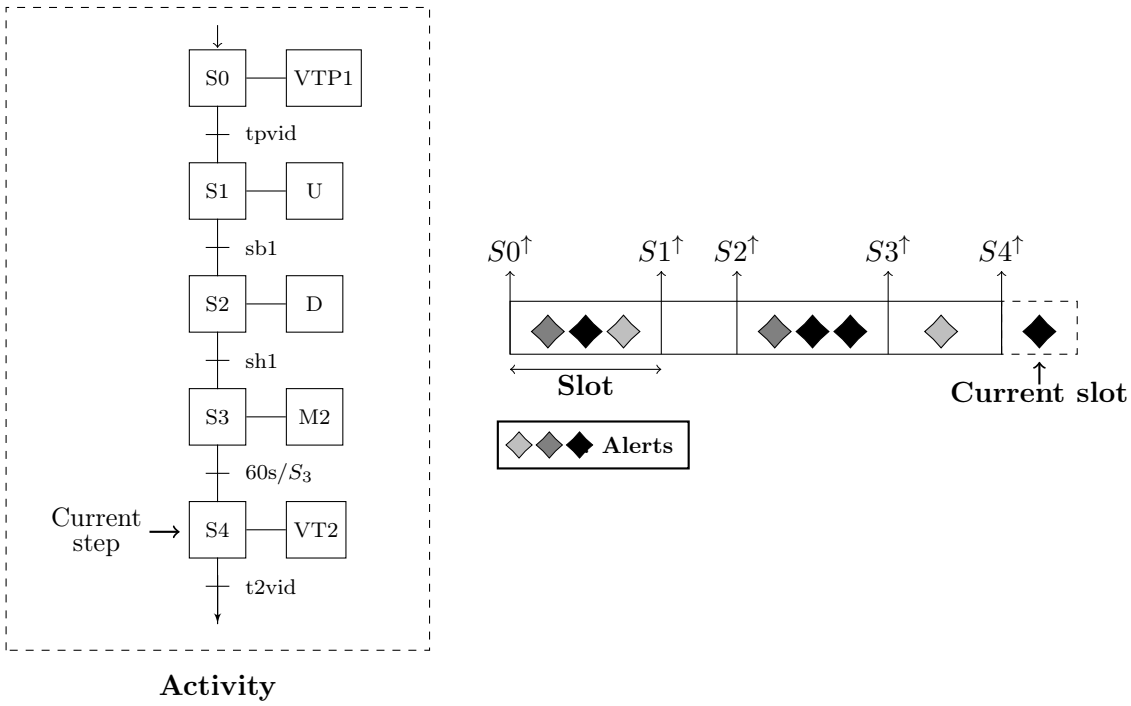


Figure 4.3: Activity-based alert window

is at step *S0* and the sensor *tpvid* is activated, control moves to step *S1*. We allocate, for each step within the activity, a *slot* in the alert window. Thus, the alert window in Figure 4.3 has 4 slots corresponding to the 4 steps in its associated activity. When control reaches a step within the activity, newly received alerts are inserted in the corresponding slot within the alert window. For instance, control in Figure 4.3 has reached step $S4$ and thus any new alert is inserted in its corresponding slot within the alert window. Through the correspondence between slots in the alert window and steps within the activity, the size of alert windows can vary dynamically depending on how long control stays within the activity's steps.

Finally, allocating slots within the alert windows allows us to introduce three alert selection policies that differ based on the number of slots taken into account:

All alerts. In this simple policy, all the alerts from the activation of the first step of the activity until the end of the activity are selected for correlation.

Alerts in adjacent steps. This policy takes into account the intermediate steps within an activity. Only alerts that occur within the last 2 slots are selected for correlation. The policy rests on the assumption that alerts that occur in adjacent slots are more pertinent than those in farther slots.

Adaptive alert selection. Instead of fixing a parameter for the number of relevant neighboring slots, this policy uses the result of previous correlations to decide on the number of past neighboring slots to include. At first, the new alert is tested against the alerts in the current slot. Then, the correlations are iteratively tested against earlier slots. If a correlation cannot be found within a slot in an iteration, no alerts in earlier slots are tested. This policy allows finding activity-long attacks by linking alerts from adjacent slots until no further correlation is possible.

4.1.4 Alert enrichment

In this section, we discuss in details our alert enrichment approach. First, we introduce a motivating example, highlight the main difficulties and discuss the limits of existing approaches in providing a solution. We then explore our enrichment model and process.

A) Motivation

Recall that the enrichment step's objective is to map alerts from IDS in the physical domain into alerts in the cyber domain so that they can be correlated with other cyber domain alerts. To illustrate the need for such a mapping, consider the attack on the example subprocess in Figure 4.2. The process monitor reports an alert A_{pm} concerning the forbidden opening ($\uparrow$) of valve *VTP1*. The alert A_{pm} is expressed using the following list of attribute key-value pairs:

$$\mathbf{A_{pm}} : [(component, VTP1), (event, \uparrow)]$$

where *component* and *event* are the attributes through which the monitor reports the attack manifestation. If the attacker opens valve *VTP1* by sending a Modbus command from an engineering workstation with address H_{ew} to the PLC with address H_{plc}, then the Modbus actuator access monitor could raise an alert A_{am}:

$$\mathbf{A_{am}} : [(source\ IP, H_{ew}), (destination\ IP, H_{plc}), (function\ code, 5), (address, 41), (data, 1)]$$

In this alert, the attributes correspond to the fields in a Modbus message (see Annex C for a presentation of Modbus).The *function code* attribute has the value 5 which refers to the modification of a single coil (1-bit variable). The value of the *address* attribute corresponds to the mapping of actuator *VTP1* in the PLC's memory. For the purpose of our example, *VTP1* is mapped to address 41 and can be accessed either through Modbus or SOAP/HTTP. Here, actuator *VTP1* is opened by setting its value to 1 through the *data* field. The protocols used to modify *VTP1* can be determined from the PLC's supported interfaces. The memory mapping of process variables, including *VTP1*, is specified during development time by an engineer.

We note that there are no common attributes between A_{pm} and A_{am}. Each alert comes from a monitor operating in a different domain, and they consequently report alerts in terms of widely different attributes. It is impossible to directly compare the alerts without a pre-processing stage. The next observation is that A_{am} is only one possible manifestation of the attack in the cyber domain. To achieve his objective, the attacker can select between different requests that command the opening of *VTP1*. These requests can be detected by the actuator access monitors but will result in alerts that vary in terms of attributes and values. For example, consider the following two possible alerts:

$$\mathbf{A'_{am}} : [(source\ IP, H_{ew}), (destination\ IP, H_{plc}), (function\ code, 15), (start\ address, 40), (quantity, 3), (values, 010)]$$

$$\mathbf{A''_{am}} : [(source\ IP, H_{ew}), (destination\ IP, H_{plc}), (function\ code, 15), (start\ address, 41), (quantity, 2), (values, 10)]$$

In this case, the attacker uses *function code* 15 which allows the modification of multiple coils at a time. The *start address*, *quantity*, and *values* give respectively the offset, the quantity of coils to be modified, and the values to write. In each of A'_{am} and A''_{am}, address 41 (corresponding to *VTP1*) is assigned the value 1. It is clear that, by further varying the *start address*, *quantity* and *values* attributes, the attacker can generate a greater variety of Modbus commands which lead to the opening of valve *VTP1*. Note also the attacker could have used another protocol such as a SOAP/HTTP request, which further increases the possibilities. The recent CrashOverride attack (Section 1.3) is a prominent example where at least four possible protocols (IEC 101, IEC 104, IEC 61850 and OPC DA) are available for the attacker.

In general, we observe that, due to the possibilities through which an operator or an attacker can interact with a PLC to affect any given process variable, the physical

domain and the cyber domain are linked through an *abstraction* relation. For instance, an actuator event such as the opening of valve *VTP1* abstracts away from the particular PLC interface (Modbus or SOAP/HTTP) or protocol variation (Modbus function code 5 or 15) that was used to modify the PLC's memory variable corresponding to *VTP1*.

One difficulty raised by this abstraction relation is how to represent the large number of cyber domain observations corresponding to a single physical domain observation. In classical alert correlation approaches, alerts are represented formally as a conjunction of attribute-value equalities. For instance, alerts A_{pm} and A_{am} above can be represented by the following logical conditions:

$$\mathbf{A_{pm}} := component = VTP1 \wedge event = \uparrow$$

$$\mathbf{A_{am}} := source\ IP = H_{ew} \wedge destination\ IP = H_{plc} \wedge function\ code = 5 \wedge address = 41 \wedge data = 1$$

Here, *component*, *event*, *source IP*, etc. are examples of typed variables describing physical and cyber domain features. Then, correlation between two alerts is typically performed by comparing their common attributes. However, this simple representation is insufficient to efficiently represent the abstraction relation between the physical domain and the cyber domain. The reason is that, using attribute-value pairs, each possible cyber domain alert corresponding to A_{pm}, such as A_{am}, A'_{am} and A''_{am} above, would need to be explicitly represented as a separate alert. This would require the correlator to generate and keep track of a significant amount of cyber domain alerts for each physical domain alert, then test correlations for each of these cyber domain alerts. What we need is an efficient way to capture and correlate all possible cyber domain alerts corresponding to a physical domain alert without having to represent separately each possible case.

To solve this issue, we take inspiration from the $\mathcal{KRA}$ model described in Section 2.3.3. The $\mathcal{KRA}$ model defines abstraction as a combination of basic operators over observations. However, the $\mathcal{KRA}$ model is primarily aimed for a bottom-up use; it does not define concretization operators for each abstraction operator. This constitutes an issue in our case because while it is always possible to map a physical process alert to the cyber domain, the opposite is not true. For instance, an unknown flow carrying IT traffic between two supervisory machines does not have any direct relation to the physical process. Thus, for efficiency reasons, we start from cyber domain alerts as abstract observations and generate the corresponding concrete observations, i.e the cyber domain alerts. This requires the availability of concretization operators.

Moreover, since observations in the $\mathcal{KRA}$ model are stored in relational databases, it is not clear how to handle the large amount of concrete observations corresponding to an abstract observation. For instance, each of the alerts $A_{am}, A'_{am}, A''_{am}, \ldots$ above would need to be inserted in a separate row of some table within the database. In the next section, we explore an enrichment model which alleviates these issues.

B) Enrichment model

From the $\mathcal{KRA}$ framework, we retain the idea of modeling an abstraction relation as a composition of elementary abstraction operators. We combine these operators to trans-

form a physical process alert into a new alert with cyber domain attributes. In particular, we treat physical process alerts as abstract observations and aim at recovering all the corresponding cyber domain concrete observations given knowledge about the ICS (protocols supported by the PLCs, protocol specifications and process variables mapping within the PLCs). In order to capture more efficiently the significant number of concrete cyber domain alerts that may correspond to a physical domain alert, we use an adequate formalism to represent an alert. We then define, for the purposes of our application, some abstraction operators along with their associated concretization operators.

Alert representation. We propose to enrich alert representations by using quantifier-free first-order logic conditions [67] expressed in one or more first-order theories such as linear integer arithmetic, strings, bit-vectors, or finite sets. These conditions are constructed using five kinds of symbols: constants, variables, functions, predicates and logical connectives ($\wedge$,$\vee$,$\Rightarrow$,$\neg$). Given a nonempty domain $\mathcal{D}$, constants refer to particular objects in $\mathcal{D}$. Variables range over the objects in $\mathcal{D}$. A function $f(x_1, x_2, \cdots, x_n)$ maps a tuple of objects $(x_1, x_2, \cdots, x_i, \cdots, x_n)$, $x_i \in \mathcal{D}$ to a single object in $\mathcal{D}$. A predicate $p(x_1, x_2, \cdots, x_n)$ maps a tuple of objects in $\mathcal{D}$ to a boolean value. A term is either a constant, a variable or a function with term arguments.

An atomic formula is a predicate symbol with term arguments. For instance, the expression $Equals(x, Plus(1, 2))$ is an atomic formula where $Equals$ is the equality predicate, $Plus$ a function with integer arguments and x is an integer variable. More complex formulae are constructed by combining atomic formulae using the logical connectives: if ϕ_1 and ϕ_2 are formulae, then $\phi_1 \wedge \phi_2$, $\phi_1 \vee \phi_2$, $\phi_1 \Rightarrow \phi_2$ and $\neg\phi_1$ are also formulae. In terms of semantics, an interpretation associates a domain object to every term and yields a truth value for a formula. For instance, in the example above, an interpretation might associate the integer 3 to the term x. Given this interpretation, $Equals(x, Plus(1, 2))$ is true. On the other hand, the interpretation which associates the integer 2 to the term x does not make the formula true. A formula is satisfiable if it has an interpretation which makes it true. Thus the example formula above is satisfiable. On the other hand, the formula $Equals(x, Plus(1, 2)) \wedge Equals(x, Plus(2, 3))$ is not satisfiable since no assignment of an integer to x can satisfy both $Equals(x, Plus(1, 2))$ and $Equals(x, Plus(2, 3))$.

Alerts are characterized by attributes which are typed variables. For an alert ψ, we use Att_ψ to denote its set of attributes. As an illustration, let us consider the case of a flow NIDS alert A_{flow}. Its set of attributes is given by $Att_{A_{flow}} = \{srcIp, dstIp, dstPort, l4p\}$. The source and destination addresses ($srcIp$ and $dstIp$) are bit-vectors of fixed size 32 (ipv4 addresses). The destination port ($dstPort$) is a 16-bit integer. The transport protocol ($l4p$) is an 8-bit integer that refers to the IANA assignment [60] (for instance, 6 for tcp and 17 for udp). The following are logical conditions which might be associated with A_{flow}:

- $\sigma_1 : srcIp = h1 \wedge dstIp = h2 \wedge dstPort = 502 \wedge l4p = 6$
- $\sigma_2 : srcIp = h1 \wedge dstIp = h3 \wedge dstPort = 80 \wedge l4p = 6$
- $\sigma_3 : srcIp \in \{h1, h3\} \wedge dstIp = h2 \wedge dstPort = 502 \wedge l4p = 6$

Here, σ_1 corresponds to a Modbus flow between $h1$ and $h2$, σ_2 represents an HTTP flow between $h1$ and $h3$, and σ_3 represents a Modbus flow for which the source address is either $h1$ or $h3$.

Using our alert representation, the cyber domain alerts corresponding to the physical domain alert A_{pm} in the previous section can be represented using the following condition $\hat{A}_{am}$:

$$\begin{cases} \hat{\mathbf{A}}_{\mathbf{am}} := \textit{source IP} = H_{ew} \wedge \textit{destination IP} = H_{plc} \wedge (\psi_1 \vee \psi_2) \\ \quad \psi_1 := \textit{unit id} = 0 \wedge \textit{protocol id} = 0 \wedge \textit{function code} = 5 \wedge \textit{address} = 41 \wedge \textit{data} = 1 \\ \quad \psi_2 := \textit{unit id} = 0 \wedge \textit{protocol id} = 0 \wedge \textit{function code} = 15 \wedge \\ \qquad\quad \textit{values}[41 - \textit{start address}] = 1 \wedge \textit{start address} \leq 41 < \textit{start address} + \textit{quantity} \end{cases}$$

Here, ψ_1 covers the straightforward case of the single coil modification (function code 5) while ψ_2 covers the case of multiple coils modifications (function code 15). In ψ_2, the *values* attribute is an array of booleans while the *start address* and *quantity* are integers. ψ_2 checks whether the address of *VTP1* falls within the range of modified coils and whether it is assigned a value of 1.

Both ψ_1 and ψ_2 above are logical conditions which represent different ways to access *VTP1* based on its *address* within the Modbus protocol. Condition ψ_1, representing a single coil modification, explicitly specifies the address of actuator *VTP1* while ψ_2 uses a start address and quantity scheme to modify multiple coils.

Abstraction operators. An abstraction operator is an application which takes as input an alert A_{con} and outputs a new alert A_{abs} such that A_{abs} is more general than A_{con}, i.e $A_{con} \Rightarrow A_{abs}$. For instance, consider an abstraction operator Op_1 which maps the flow NIDS alert $A_{am} : \textit{srcIp} = h1 \wedge \textit{dstIp} = h2 \wedge \textit{dstPort} = 502 \wedge \textit{l4p} = 6$ into a new alert $Op_1(A_{am}) : \textit{dstIp} = h2 \wedge \textit{dstPort} = 502 \wedge \textit{l4p} = 6$. Compared with A_{am}, the new alert $Op_1(A_{am})$ does not specify the source of the anomalous flow (attribute *srcIp*). Due to this loss of information with regard to the source address, A_{am} can be considered as a special case of $Op_1(A_{am})$ where $\textit{srcIp} = h1$. In other words, $A_{am} \Rightarrow Op_1(A_{am})$.

Looking at the relation between attributes at the cyber and at the physical domains, we can distinguish between two cases. On one hand, some cyber domain attributes do not have any bearing on the physical domain. For instance, the source network address of a Modbus command does not tell us anything about which PLC is queried or which actuator will be impacted. Since such attributes are irrelevant, they can simply be hidden when performing the mapping. On the other hand, attributes such as the destination network address, the function code, and the process variable address indicate the target PLC and actuator. We now introduce two abstraction operators that correspond to these cases.

Attribute hiding abstraction operator. An attribute hiding operator H_{attr} generates abstract alerts by removing constraints over an attribute *attr* in the input alert.

The abstraction operator Op_1 above is an example of an attribute hiding operator. The main use of this operator is to discard attributes which are not relevant for the enrichment process.

The operator generates $A_{abs} = H_{attr}(A_{con})$ from input alert A_{con} by: (i) removing $attr$ from $Attr_{A_{con}}$, i.e $Attr_{A_{abs}} = Attr_{A_{con}} \setminus \{attr\}$, and (ii) by *forgetting* any constraints regarding $attr$. Several methods have been studied to compute this forgetting [98]. The approach we follow consists in removing all the constraints associated with $attr$ from the conjunctions in the disjunctive normal form (DNF) of the A_{con}. Through the DNF, A_{con} can be rewritten as:

$$\bigvee_{k=1}^{m} \Big(\bigwedge_{a \in Att_{A_{con}}} \lambda_k[a] \Big)$$

where $\lambda_k[a]$ is a logical formula containing only constraints associated with the attribute $a \in Att_{A_{con}}$. Then:

$$H_{attr}(A_{con}) = \bigvee_{k=1}^{m} \Big(\bigwedge_{a \in Att_{A_{con}} \setminus \{attr\}} \lambda_k[a] \Big)$$

.

This procedure guarantees that $A_{con} \Rightarrow H_{attr}(A_{con})$, since by the rule of conjunction elimination[1]:

$$\bigwedge_{a \in Att_{A_{con}}} \lambda_k[a] \Rightarrow \bigwedge_{a \in Att_{A_{con}} \setminus \{attr\}} \lambda_k[a]$$

and thus by the rule of constructive dilemma[2]:

$$\bigvee_{k=1}^{m} \Big(\bigwedge_{a \in Att_{A_{con}}} \lambda_k[a] \Big) \Rightarrow \bigvee_{k=1}^{m} \Big(\bigwedge_{a \in Att_{A_{con}} \setminus \{attr\}} \lambda_k[a] \Big)$$

which means that:

$$A_{con} \Rightarrow H_{attr}(A_{con})$$

Finally, note that several attributes can be hidden simply be a composition of elementary attribute hiding operators.

Attributes equivalence abstraction operator. While an attribute hiding operator completely eliminates an unneeded attribute, an equivalence operator replaces constraints over some attributes of interest in the input alert with new constraints in the abstract alert.

For instance, consider the flow NIDS alert $A_{con} : srcIp \in \{h1, h2\} \wedge dstIp = h3 \wedge dstPort = 502 \wedge l4p = 6$. In general, flows within an ICS can be assigned to zones

[1] $(p \wedge q) \Rightarrow p$

[2] $((p \Rightarrow q) \wedge (r \Rightarrow s) \wedge (p \vee r)) \Rightarrow (q \vee s)$

depending on the location of the flow's endpoints. Suppose that addresses $h1$, $h2$, and $h3$ belong to $H_{Control}$ which denotes the set of network addresses within the control zone (for instance, belonging to PLCs or local HMIs). In this case, flow NIDS alerts satisfying $srcIp \in H_{Control} \wedge dstIp \in H_{Control}$ would represent horizontal flows among different nodes within the control zone.

Let's use a new attribute, $flowZone \in \{Control, Supervisory, Hybrid\}$, to describe the zone of a flow which can be either within the control and supervisory zones or crossing different zones (hybrid). Thus, we have the following equivalence relation:

$$srcIp \in H_{Control} \wedge dstIp \in H_{Control} \Leftrightarrow flowZone = \text{Control}$$

Given this equivalence relation, an equivalence abstraction operator would transform A_{con} into an alert A_{abs} where:

$$Attr_{A_{abs}} = (Attr_{A_{con}} \setminus \{srcIp, dstIp\}) \cup \{flowZone\}$$

and:

$$A_{abs} : flowZone = Control \wedge dstPort = 502 \wedge l4p = 6$$

We make the following observations on this example. First, alert A_{abs} is more generic than alert A_{con} since the former reports an unknown Modbus flow within the control zone without specifying any specific Ip addresses, while A_{con} is restricted to Modbus flows from either $h1$ and $h2$ to $h3$. Secondly, in the example, all hosts $h1$, $h2$ and $h3$ belong to the same zone. This need not be the case in general. For instance, if $h2$ corresponds to an engineering workstation at the supervisory level, then A_{con} would include flows from both the Control and Hybrid categories. In this case, A_{con} would be mapped to:

$$A_{abs} : (flowZone = Control \vee flowZone = Hybrid) \wedge dstPort = 502 \wedge l4p = 6$$

We now generalize the above procedure. Let Π be a bijective application that associates conditions on some subset of the concrete alert's attributes (denoted by $Att_{Dom(\Pi)}$) to equivalent conditions on the new abstract alert's attributes (denoted by $Att_{Range(\Pi)}$). For instance:

$$Att_{Dom(\Pi)} = \{srcIp,\ dstIp\}, Att_{Range(\Pi)} = \{flowZone\}$$

$$srcIp \in H_{Control} \wedge dstIp \in H_{Control} \xrightarrow{\Pi} flowZone = \text{Control}$$

$$srcIp \in H_{Supervisory} \wedge dstIp \in H_{Supervisory} \xrightarrow{\Pi} flowZone = \text{Supervisory}$$

Let E_Π be the equivalence abstraction operator based on Π. To generate $A_{abs} = E_\Pi(A_{con})$ from input alert A_{con}, we make use of the DNF form of A_{con}: $\bigvee_{k=1}^{m} (\bigwedge_{a \in Att_{A_{con}}} \lambda_k[a])$. We find, for every conjunction $\sigma_k : \bigwedge_{a \in Att_{A_{con}}} \lambda_k[a]$, the set:

$$C_k = \{\Pi(\varphi_{con}) \mid \varphi_{con} \in Dom(\Pi) \wedge \bigwedge_{a \in Att_{A_{con}}} \lambda_k[a] \Rightarrow \varphi_{con}\}$$

representing the conditions in the range of Π which are compatible with σ_k. Then, σ_k is replaced with:

$$\bigwedge_{a \in Att_{A_{con}} \setminus Att_{Dom(\Pi)}} (\lambda_k[a]) \wedge \bigvee_{c \in C_k} c$$

and used to construct the abstract alert:

$$\begin{cases} A_{abs} = E_\Pi(A_{con}) = \bigvee_{k=1}^{m} (\bigwedge_{a \in Att_{A_{con}} \setminus Att_{Dom(\Pi)}} (\lambda_k[a]) \wedge \bigvee_{c \in C_k} c) \\ Att_{A_{abs}} = (Att_{A_{con}} \setminus Att_{Dom(\Pi)}) \cup Att_{Range(\Pi)} \end{cases}$$

This procedure guarantees that $A_{con} \Rightarrow E_\Pi(A_{con})$. To see this, note that by the definition of C_k, we know that: $\forall c \in C_k$, $\bigwedge_{a \in Att} \lambda_k[a] \Rightarrow c$. Thus:

$$\bigwedge_{a \in Att} \lambda_k[a] \Rightarrow \bigvee_{c \in C_k} c$$

Moreover, we also know by the rule of conjunction elimination that:

$$\bigwedge_{a \in Att_{A_{con}}} \lambda_k[a] \Rightarrow \bigwedge_{a \in Att_{A_{con}} \setminus Att_{Dom(\Pi)}} \lambda_k[a]$$

Thus:

$$\bigwedge_{a \in Att_{A_{con}}} \lambda_k[a] \Rightarrow \bigwedge_{a \in Att_{A_{con}} \setminus Att_{Dom(\Pi)}} (\lambda_k[a]) \wedge \bigvee_{c \in C_k} c$$

Finally, using the rule of constructive dilemma, we have:

$$\bigvee_{k=1}^{m} (\bigwedge_{a \in Att_{A_{con}}} \lambda_k[a]) \Rightarrow \bigvee_{k=1}^{m} (\bigwedge_{a \in Att_{A_{con}} \setminus Att_{Dom(\Pi)}} (\lambda_k[a]) \wedge \bigvee_{c \in C_k} c)$$

and thus:

$$A_{con} \Rightarrow E_\Pi(A_{con})$$

Concretization operators. For every abstraction operator Op, we can also define an inverse operator Op^{-1} called a concretization operator. Given an alert A_{abs}, the concretization operator outputs an alert A_{con} which subsumes all the alerts that could have been mapped to A_{abs} by Op. Formally, this means that Op^{-1} outputs an alert A_{con} such that for all alerts A'_{con}, if $Op(A'_{con}) = A_{abst}$, then $A'_{con} \Rightarrow A_{con}$.

Attribute hiding concretization operator. Consider the attribute hiding abstraction operator H_{srcIp} which hides the *srcIp* attribute for flow NIDS alerts. Let $A_{abs} : dstIp = h2 \wedge dstPort = 502 \wedge l4p = 6$ be an abstract alert which is the result of applying H_{scrIp} on some concrete alert A_{con}. Possible conditions for A_{con} include : $srcIp = h1 \wedge dstIp = h2 \wedge dstPort = 502 \wedge l4p = 6$ or $srcIp \in \{h1, h3, h4\} \wedge dstIp = h2 \wedge dstPort = 502 \wedge l4p = 6$. Both of these conditions are mapped to A_{abs} by H_{srcIp}. In fact, any constraint on the *srcIp* would have been hidden by H_{srcIp}.

Thus, to cover all cases, the concrete alert need not specify any constraint on *srcIp* so that *srcIp* can range over all its possible values. Thus, $H^{-1}_{srcIp}(A_{abs}) = A_{abs}$.

We now show that $H^{-1}_{srcIp}(A_{abs})$ effectively subsumes all possible concrete alerts that can be mapped to A_{abs} by H_{srcIp}. In general, for any attribute hiding abstraction operator H_{srcIp}, its corresponding concretization operator H^{-1}_{srcIp} associates for every abstract alert A_{abs} a concrete alert $A_{con} = A_{abs}$. Moreover, given any other concrete alert A'_{con} such that $H_{srcIp}(A'_{con}) = A_{abs}$, we know that $A'_{con} \Rightarrow A_{abs}$ (by definition of an abstraction operator). And since $A_{con} = A_{abs}$, then $A'_{con} \Rightarrow A_{con}$ which means that $H^{-1}_{srcIp}(A_{abs})$ effectively subsumes all possible concrete alerts that can be mapped to A_{abs} by H_{srcIp} as required.

Attributes equivalence concretization operator. Consider the attributes equivalence abstraction operator E_Π from the zone example above. Let $A_{abs} : flowZone = control \wedge dstport = 502 \wedge l4p = 6$ be an abstract alert which is the result of applying E_Π on some concrete alert A_{con}.

Since A_{abs} specifies that $flowZone = Control$, A_{con} must refer to flows within the control zone. Thus, A_{con} must satisfy the equivalent constraint given by Π:

$$srcIp \in H_{Control} \wedge dstIp \in H_{Control}$$

Moreover, since no information remains in A_{abs} as to the specific source and destination network addresses that were specified by A_{con} except that they belong to the control zone, we get that:

$$E^{-1}_\Pi(A_{abs}) : srcIp \in H_{Control} \wedge dstIp \in H_{Control} \wedge dstport = 502 \wedge l4p = 6$$

We can show that $E^{-1}_\Pi(A_{abs})$ above subsumes all possible concrete alerts which can be mapped to A_{abs} by E_Π. Since E^{-1}_Π replaces the constraint $flowZone = Control$ with the equivalent constraint $srcIp \in H_{Control} \wedge dstIp \in H_{Control}$, we get that:

$$E^{-1}_\Pi(A_{abs}) \Leftrightarrow A_{abs}$$

And since for any other concrete alert A'_{con} such that $E_\Pi(A'_{cont}) = A_{abs}$ we know that $A'_{con} \Rightarrow A_{abs}$ (by definition of an abstraction operator), then we get that:

$$A'_{con} \Rightarrow E^{-1}_\Pi(A_{abs})$$

which means that $E^{-1}_\Pi(A_{abs})$ subsumes all possible concrete alerts which can be mapped to A_{abs} by E_Π as required.

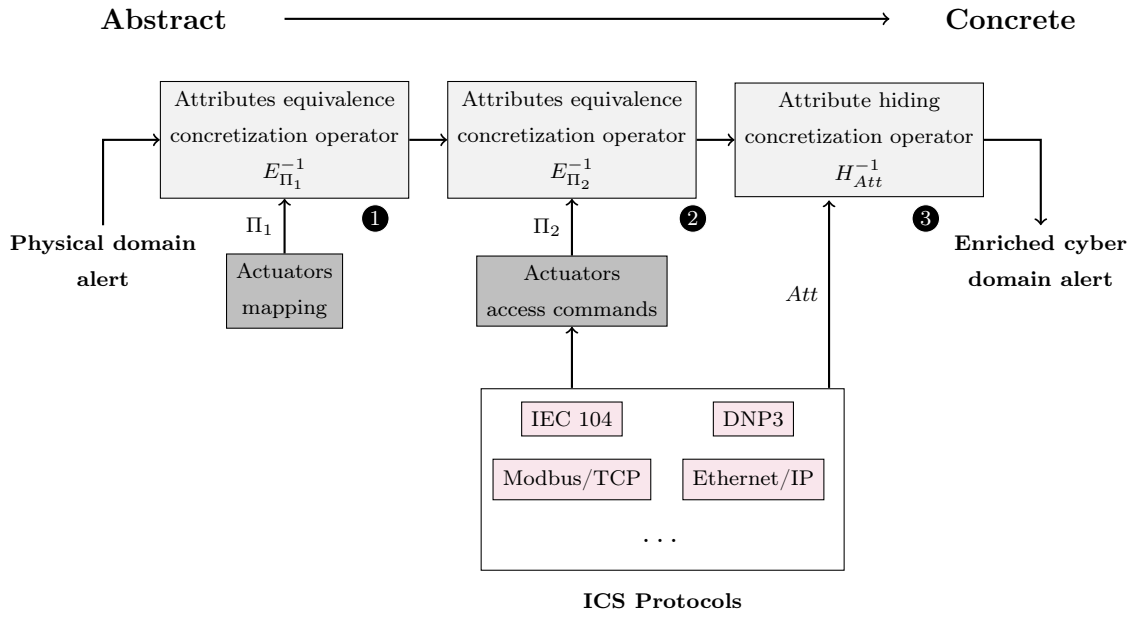


Figure 4.4: Overview of the alert enrichment process

Finally, note that the swapping operation used above to generate concrete alert $E_{\Pi}^{-1}(A_{abs})$ can also be performed through $E_{\Pi^{-1}}(A_{abs})$ (recall that Π is a bijective application and thus has an inverse Π^{-1}).

C) Alert enrichment process

The enrichment process is depicted in Figure 4.4. The process takes as input a physical process domain alert, expressed in terms of process domain attributes, and outputs an enriched cyber domain alert expressed in terms of cyber domain attributes. The heart of the enrichment consists of the successive application of three concretization operators: two attributes equivalence concretization operators (❶, ❷) and one attributes hiding concretization operator (❸).

Each concretization operator uses information from the ICS to perform its mapping. We now discuss in details each of the three concretization operators. To illustrate the enrichment process, we take as an example the following process-domain alert:

$$\mathbf{A_{pm}} := \mathit{component} = \mathit{VTP1} \wedge \mathit{event} = \uparrow$$

Equivalence concretization operator $E_{\Pi_1}^{-1}$ (❶)

The first step in the alert enrichment process is to map each actuator to (i) the set of network addresses of the PLCs which control it, (ii) the protocols which can be used to access the actuator on each PLC identified in (i), and (iii) the address or identifier which is associated with the actuator for each protocol and PLC identified in (ii). Table 4.1

Table 4.1: Example of actuators mapping information

Actuator	**PLC Ip address**	**Protocol**	**Address/identifier**
VTP1	H_{PLC}	Modbus/TCP	41
		DNP3	97

shows an example of such a mapping for actuator VTP1. In this example, VTP1 can be accessed by using either Modbus/TCP with address 41 or DNP3 with address 97 on PLC with network address H_{PLC}.

This mapping is performed through an attributes equivalence concretization operator $E^{-1}_{\Pi_1}$ where Π_1 associates each actuator with information (i), (ii) and (iii) above. Recall that Π_1 is an application which associates equivalent constraints between an abstract alert and a concrete alert. For instance, in the case of actuator $VTP1$ in Table 4.1, Π_1 would associate the concrete level constraint:

$$dstIp = H_{PLC} \wedge ((protocol = \text{Modbus} \wedge address = 41) \vee (protocol = \text{DNP3} \wedge address = 97))$$

with the abstract level constraint:

$$component = VTP1$$

When the concretization operator $E^{-1}_{\Pi_1}$ is applied to alert A_{pm} we obtain:

$$\mathbf{E^{-1}_{\Pi_1}(A_{pm})} := event = \uparrow \wedge (dstIp = H_{PLC} \wedge ((protocol = \text{Modbus} \wedge address = 41) \vee (protocol = \text{DNP3} \wedge address = 97)))$$

Obtaining the information necessary to construct Π_1 is straightforward if adequate device inventory databases are maintained at the ICS plant. While ICS plants are encouraged to maintain such databases with regular updates and proper security mechanisms [115], their availability is not guaranteed in general. In this case, alternative ways to gather this information must be sought. The authors in [49] discuss several data sources which can be used in order to retrieve the mapping between process variables and memory locations. For instance, project files used to program PLCs can contain valuable information such as the presence of descriptive text and friendly tags associated with memory locations. In many cases, these project files can be downloaded from PLCs and reviewed using official vendor software. A drawback of this data source is the need to use adequate software for every PLC constructor. Another possible data source resides in local HMIs and workstations which allow users to monitor and interact with the physical process. As in the case of PLC project files, HMIs configuration files can be viewed using official vendor-specific software. Alternatively, if HMI configuration files are not available, the compiled runtime files gathered from HMIs can be analyzed using HEX editors to extract useful information. However, this task can be time-consuming

and effort prone. Finally, network traffic can also be a useful source of information by using safe ICS asset discovery techniques such as passive monitoring [143]. In practice, collating information from many of these data sources might be necessary to retrieve the required information.

Equivalence concretization operator $E^{-1}_{\Pi_2}$ (❷)

The second step of the enrichment process is to associate events on actuators with ICS protocol commands. For instance, to cause a rising edge (↑) of discrete actuator $VTP1$ using Modbus, an attacker can use either function code 5 (writing a single coil) or 15 (writing multiple coils). This mapping is performed through an attributes equivalence concretization operator $E^{-1}_{\Pi_2}$.

To define Π_2, we propose to characterize actuators access commands using three features: (i) the *operation type*, (ii) the *variable type*, and (iii) the *variable access mode*. We discuss in details each of these features through four ICS protocols : Modbus/TCP, DNP3, IEC-104, and Ethernet/IP. Details about each of these protocols can be found in Annex C. We choose to focus on these particular protocols because of their popularity [103], the availability of their specifications, and the differences in their design and domains of applicability.

- **Operation type.** This corresponds to a type of operation to be performed on the actuator such as a *read* or a *write* operation. All the above protocols allocate a specific field to characterize the operation: function code (Modbus, DNP3), type identification (IEC-104) or request service (Ethernet/IP). An operation can either characterize the type of variable on which it operates (for instance, function code 5 in Modbus and type identification 45 in IEC-104 for discrete outputs), or remain agnostic as to the type of variable manipulated (for example, both DNP3 and Ethernet/IP define generic write operations: function code 0x02 and Write Data request service respectively).

- **Variable type.** As discussed above, in Ethernet/IP and DNP3, operations are not specific to a particular type of variables. DNP3 uses the combination of the group and variation fields to specify the type of the target variables. On the other hand, the type targeted by Ethernet/IP is implied by the class identifier of the object's path (for example, Discrete Output Point with class identifier 0x09). Thus, we distinguish on one hand between protocols where the type of variables is either implied by the operation (Modbus, IEC-104) or by the variable's access location (Ethernet/IP), and on the other hand protocols where the type of target variables is given by explicit fields (group, variation or class).

- **Variable access mode.** To refer to variables, Modbus, IEC-104 and DNP3 use addresses while Ethernet/IP use either combination of class, object and attribute identifiers or symbolic identifiers. Thus, we distinguish between address-based and identifier-based variable access modes. Protocols such as Modbus, IEC-104, and

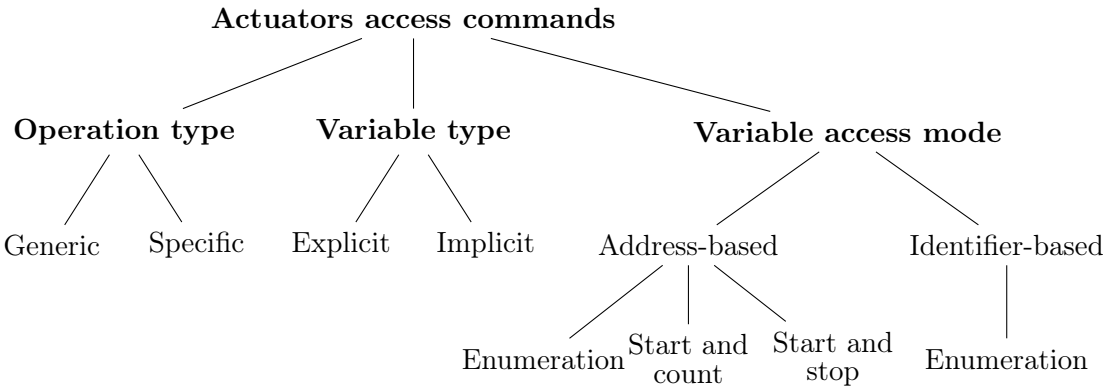


Figure 4.5: Main features of actuator access commands for common ICS protocols

DNP3 that use address-based access modes can refer to variables either using explicit ranges (start address and end address), start address and count, or by enumerating the addresses. Protocols that use identifier-based access modes such as Ethernet/IP enumerate identifiers (sets of request paths).

Figure 4.5 provides a classification of actuator access commands based on the preceding discussion. We note that many ICS protocols tend to share similar schemes to access actuators. For instance, Modbus, IEC-104 and DNP3 all use address-based variable access modes. Thus, the same equivalence relations that are used for Modbus can be reused for IEC-104 and DNP3 provided attribute names are adapted to each protocol (for example, while Modbus and DNP3 use *function codes*, IEC-104 uses *type identifications*). Modbus uses specific operation types with implicit variable type and address-based variable access modes. Π_2 would then associate the concrete level constraint:

$$(function\ code = 5 \wedge address = c \wedge data = 1) \vee (function\ code = 15 \wedge$$
$$values[c - start\ address] = 1 \wedge start\ address \leq c < start\ address + quantity)$$

with the abstract level constraint:

$$event = \uparrow \wedge address = c \wedge protocol = \text{Modbus}$$

Here, c is a constant integer, *function code* refers to the explicit operation type and the address-based variable access modes (enumeration and start and quantity) are represented by the following two conditions:

$$address = c \wedge data = 1$$

$$values[c - start\ address] = 1 \wedge start\ address \leq c < start\ address + quantity$$

Both of the above conditions can be used to express address-based variable access modes for other protocols that use this access mode.

Note that the enrichment model is generic and expressive enough in order to handle cases which are not covered by the classification in Figure 4.5. Information about where a protocol lies within the classification can be manually extracted from the protocol specifications of each supported protocol. For most of the commonly used ICS protocols [103] (including Ethernet/IP, Profinet, EtherCAT, Modbus/TCP and DNP3), such specifications are available. In the case of proprietary protocols such as Siemens S7, reverse engineering [102] techniques can be used.

Attributes hiding concretization operator H_{Att}^{-1} (❸)

The last concretization operator is an attributes hiding concretization operator which handles all the cyber domain attributes that are irrelevant to the physical domain. These attributes include both protocol-specific attributes such as the *transaction id* attribute for Modbus and more general cyber domain attributes such as the source network address.

4.1.5 Alert aggregation

To evaluate our solution within the overall correlation process, we feed the output of our alert enrichment approach to an alert aggregation stage inspired from existing similarity-based alert correlation approaches [63, 139]. The alert aggregation phase decides, for each new alert A_{new} and a set of previous alerts $\mathcal{A}_{old}$ from the alert selection stage, the subset of $\mathcal{A}_{old}$ which are correlated with A_{new} based on the similarity of their attributes. The decision is performed for each pair of (enriched) alerts (A_{new}, A_{old}), $A_{old} \in \mathcal{A}_{old}$. On the basis of these pairwise correlations, the correlator constructs meta-alerts (clusters of alerts).

Multiple strategies can be followed to generate meta-alerts from pairwise correlations. One possible strategy is to enforce that alerts belonging to the same cluster must all be pairwise correlated pairwise. For instance, let $S = \{A_1, A_2, A_3\}$ be a set of alerts and *corr* a correlation relation such that $corr(A_i, A_j)$ if and only if A_i and A_j are correlated. Then, using this strategy, S is a meta-alert if and only if $corr(A_1, A_2)$, $corr(A_2, A_3)$, and $corr(A_1, A_3)$. This strategy yields tight clusters where all the alerts are correlated to one another. The drawback of this strategy is the possibility of missing interesting correlations.

For instance, consider the case of an attacker who attempts to wear a valve, i.e by rapidly opening and closing it. Furthermore, suppose that the attacker can impact the valve through two different protocols. As part of this attack, the attacker uses the first protocol to open the valve and the second protocol to close it. Moreover, the attacker issues these commands from multiple engineering stations which normally should not issue commands to PLCs. Suppose that this yields four alerts: two physical domain alerts A_{phy}^1, A_{phy}^2 relative respectively to the forbidden opening and closing of the valve, and two cyber domain alerts A_{cy}^1, A_{cy}^2 relative respectively to the issuing of commands from different engineering stations using different protocols. Then we would have $corr(A_{phy}^1, A_{cy}^1)$ (forbidden command to open the valve and physical domain violation reporting the opening of the valve) ,$corr(A_{phy}^1, A_{phy}^2)$ (two physical domain

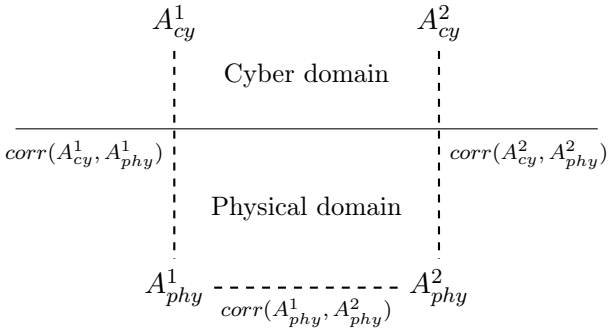


Figure 4.6: Indirect correlation between alerts

violations on the same valve), and $corr(A^2_{phy}, A^2_{cy})$ (forbidden command to close the valve and physical domain violation reporting the closing of the valve). However, A^1_{cy} and A^2_{cy} are not directly correlated since they report commands issued using different protocols from different stations. This implies that, following this strategy, the alerts $\{A^1_{phy}, A^2_{phy}, A^1_{cy}, A^2_{cy}\}$ would not form a meta-alert, even though they are part of the same attack.

In the example above, A^1_{cy} and A^2_{cy} are *indirectly* correlated because their correlation can be understood only through the effect they have on the physical process, as reported by the alerts A^1_{phy} and A^2_{phy}. Thus another strategy is to allow two alerts A_i and A_j to belong to the same meta-alert if either A_i, A_j are correlated, or if there exists an alert A_k such that A_i, A_k and A_k, A_j are respectively correlated. Thus, S above would constitute a meta-alert if $corr(A_1, A_2)$ and $corr(A_2, A_3)$. In this case, it is not necessary for A^1_{cy} and A^2_{cy} to be pairwise correlated in order to end up in the same cluster. Thus, this strategy is looser than the first and can yield meta-alerts containing very different alerts. We argue that such a strategy is the most promising in our use case because of the possibility that two alerts which are indirectly correlated in one domain might be directly correlated with alerts which are directly correlated in another domain as illustrated in Figure 4.6.

To decide whether A_{new} and A_{old} are correlated, we test, using an SMT solver, whether there is an assignment to all or a subset of their common attributes which makes $A_{new} \wedge A_{old}$ true. However, this constraint can be quite strong. For instance, if two Modbus payload-based IDS alerts A_1 and A_2 report the same source, same destination, and are both commands to override coils albeit with different function codes (5 and 15), we would still like to fuse them since they might be symptomatic of an attacker who uses different function codes for consecutive commands. Thus a weaker yet still interesting constraint might be:

$$A_1.source\ IP = A_2.source\ IP \wedge A_1.dest.\ IP = A_2.dest.\ IP$$
$$\wedge\ A_1.function\ code \in WFC \wedge A_2.function\ code \in WFC$$

where WFC is the set of Modbus *writing* function codes.

In general, weaker constraints are obtained by either : (i) partitioning the set of values of a common attribute and checking whether the attribute's values in the pair of tested alerts belong to the same partition, (ii) ignoring a common attribute. Examples of the first type of weak constraint include checking whether source/destination addresses belong to the same network zone (control, supervisory, etc.) or whether the function codes refer to the same type of operations on process variables (write, read, etc.). Information about the network's topology can be obtained either through the plant's asset inventory databases or using passive asset discovery techniques [143], while the operation classes specific to each protocol can be gathered from the protocol specifications if available or through reverse engineering [102].

By default, the correlation is tested using the strongest constraint (equality between all common attributes). If the correlation fails, we test with increasingly weaker constraints by applying (i) then (ii) on the attributes shared by the alerts that are compared. Note that the number of weak constraints that are tested is bounded since the set of common attributes is finite and the number of ways a constraint on each attribute can be weakened is finite (using either (i) or (ii) above). To avoid too weak correlations, the user can adjust two parameters : (a) the maximum number of tested weak constraints, (b) a list of attributes for which weaker constraints cannot be produced using either (i) or (ii) above. Weak correlations are tested using a breadth-first search strategy starting from the strongest possible constraint and stopping the search when no more constraint satisfying both (a) and (b) can be further produced. The type of correlation (strong, weak through partitioning, weak through hiding) is specified in the alert correlations received by the security operator.

4.2 Evaluation

To evaluate our approach, we need a sufficiently complex ICS. However, as discussed in Section 3.3 due to privacy concerns, real data from operational plants is hard to obtain. As a substitute, we develop a testbed with a complex physical process and realistic ICS architecture. Compared to most experimental setups with publicly available datasets that can be found in the literature, our testbed is either comparable[3] or significantly more complex[4].

Our evaluation testbed consists of a hardware-in-the-loop setting including a simulation of the physical process shown in Section 3.3, Figure 3.11. An ICS architecture with both control and supervisory levels, shown in Figure 4.7, steers the physical process. The architecture involves real components found in operational ICS (i.e, PLCs, control servers, HMI, etc.). Control is distributed using three PLCs (Schneider M340, Schneider M580, and Wago IPC-C6 with additional RTU 750-873). Control logics are implemented in SFC. The Schneider M340 and Wago IPC-C6 PLCs support Modbus, while the Schneider M580 PLC supports both Modbus and SOAP/HTTP. At the supervisory level, an OPC-UA server polls data and relay commands to the PLCs. A supervisor HMI

[3]https://itrust.sutd.edu.sg/dataset

[4]https://sites.google.com/a/uah.edu/tommy-morris-uah/ics-data-sets

provides a global view of the state of the physical process to the operators. Engineering workstations allow engineers to change the control logics of the PLCs. Operators are not allowed to perform manual operations from the engineering workstations.

Figure 4.7 also depicts the IDS placement within the ICS. The network flow IDS [5] covers the whole ICS traffic through a mirror port at the level of the switch. The payload-based IDS [149] (PB1, PB2, and PB3) monitor the ICS traffic (Modbus/TCP and SOAP/HTTP) reaching each PLC. Finally, the physical process IDS (see Chapter 3) (PP1, PP2, and PP3) operate on the link between each PLC and its local HMI. For practical purposes, we monitor the actuators/sensors signals on each PLC using the supervisory HMI traffic. However, the physical process IDS could also directly monitor the communications between the sensors/actuators and the PLC using an adequate tap.

4.2.1 Attacks and operators interventions

To evaluate our approach, we need to generate both attack and legitimate traffic. We follow the same approach discussed in Section 3.3.2 to generate attacks and operators interventions. Additionally, to reflect the possibilities afforded to an attacker when carrying attacks from the cyber domain, we specify two dimensions along which the attacks can vary : (i) the source of the attack, and (ii) the protocols used to carry the attack. The source of the attack can either be a host which is allowed to send commands to PLCs (such as an HMI or an OPC server), or a host which is not used for process operations (such as an engineering workstation). The attacker can alternate between several protocols (SOAP/HTTP or Modbus) and use different operations within each protocol (Modbus with function codes 5 or 15 for instance). When carrying attacks, both the source of the attacks and the protocols used are randomly selected among all possibilities afforded to the attacker.

4.2.2 Implementation

The IDS are implemented in C++/Python following their descriptions in the original papers [5, 149]. We use Argus [3] to recognize network flows and rely on Bro [18] to extract Modbus/TCP data from network traffic. The correlator is implemented in Python. To test for satisfiability when applying the mapping rules to alerts, the correlator queries CVC4 [26], an SMT solver, through the PySMT library [120]. The communication between PySMT and the SMT solver uses the SMTLib standard. Thus, any SMT solver supporting the standard and implementing the required theories can be used. To carry interventions, we use either : (i) the Modbus client module in Metasploit [93] which is queried through its RPC interface, (ii) the OPC-UA server by means of an OPC-UA client [109], or (iii) SOAP/HTTP requests by means of a SOAP client [150]. All our correlation performance analysis is performed on an Intel Dual Core i7 2.6 Ghz machine with 16 GB of RAM running Linux kernel 4.4.0.

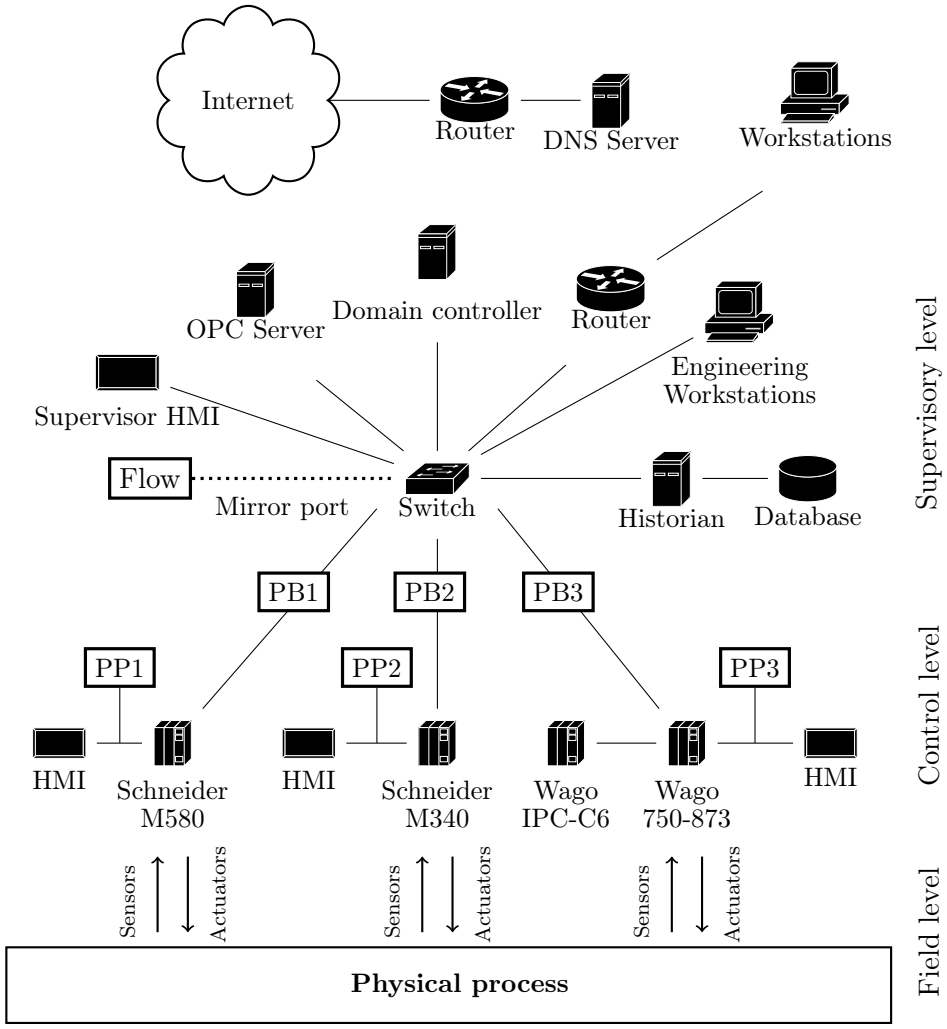


Figure 4.7: Simplified ICS architecture

4.2.3 Datasets

We base our evaluation on 5 network captures spanning a total of 62 hours. To train the IDS base profiles, we use a 14 hours capture free of any malicious actions and containing 27 legitimate operator interventions. This dataset reflects realistic conditions where certain legitimate behaviors are absent due to the limited training window. The number of activities' executions in the training data ranges from 28 to 155 cycles. We also generate 4 network captures which contain attacks as described in Section 4.2.1. These network captures span a total of 48 hours during which 26 attacks and 98 legitimate operator interventions are carried. The main différence with respect to the dataset used to evaluate our intrusion detection approach in Section 3.3, is the possibility for the attackers and operators to use multiple protocols in order to carry out their tasks. Whereas the dataset in Section 3.3 contains only Modbus traffic, the dataset used to evaluate correlation uses Modbus, SOAP/HTTP, and OPC UA to access process variables. As in the case of the intrusion detection dataset, the correlation evaluation dataset is publicly available. All datasets will be available online[5].

4.2.4 Alert aggregation parameters

Recall from Section 4.1.5 that computing weak correlations involves either (i) partitioning the set of values of a common attribute, or (ii) ignoring a common attribute. In our experiments, we partition the values of three category of attributes. IP addresses are partitioned depending on the zones of the ICS (supervisory and control zones), function codes are partitioned depending on the type of operations they perform on process variables (either reading or writing operations), and actuators/sensors (*component* attribute) are partitioned depending on the PLC which control them (components that are controlled by the same PLC are grouped together). All attributes can be ignored except for the *component* attribute in the physical domain. This is because ignoring the *component* attribute would mean that physical domain alerts are only correlated based on the events they report (rising or falling edge) which is insufficient to yield good correlation results.

4.3 Analysis

In this section, we present and analyze the results of our approach on the traces described in Section 4.2.3. We first describe the metrics we use for the evaluation and compare the results against a classical temporal window correlation approach [140].

Metrics Our correlator links two alerts which are considered to be manifestations of an operator or attacker's intervention (i.e, sequence of commands). Correlation errors happen either when two alerts which are not from the same intervention are linked together (*incorrect correlation*), or when two alerts which are from the same intervention

[5] https://persyval-platform.univ-grenoble-alpes.fr/0/searchbyrecently

are not linked (*missing correlation*). To evaluate our correlation approach with respect to these possible errors, we use the following two classical correlation metrics [104] :

$$\textit{False correlation rate} = \frac{\#\ \textit{incorrect correlations}}{\#\ \textit{produced correlations}}$$

$$\textit{Missing correlation rate} = \frac{\#\ \textit{missing correlations}}{\#\ \textit{expected correlations}}$$

Here, *produced correlations* refer to all the correlations that were established by the correlator. These *produced correlations* can include both correct and incorrect ones. *Expected correlations* refer to the correlations that should be established by a correct correlator. Thus, *expected correlations* include both the correlations that were correctly established by the correlator, in addition to the correct correlations missed by the correlator. The *false correlation rate (FCR)* reflects the proportion of incorrect correlations in the correlations produced by the correlator. The *missing correlation rate (MCR)* represents the proportion of missing correlations in the number of correlations expected to be produced by the correlator given the alerts. We identify the expected correlations by manually analyzing the traces since we have access to detailed information (time, protocols, target actuators, etc.) about the attacks and operator interventions carried in the evaluation traces. Using this information, we can determine which alerts must be correlated in order to correspond to the manifestations of an attack or of an operator's intervention.

In addition, we also evaluate the reduction in the number of alerts which are sent to the operator after correlation. This reduction is determined through the following metric :

$$\textit{Reduction} = 1 - \frac{\#\textit{meta-alerts} + \#\textit{uncorrelated alerts}}{\#\textit{total alerts}}$$

While higher values of the reduction metric mean fewer alerts to handle for the security operators, the reduction should still allow distinguishing between different operator/attacker interventions, i.e by keeping a low FCR value.

Results To evaluate our approach, we compute the FCR, MCR and reduction metrics on the traces described in Section 4.2.3. We compare our results with a fixed-size sliding window correlator. This correlator keeps a fixed size window of old alerts and links each new alert with all alerts in the window. We compute the metrics for different sizes of the temporal window. Tables 4.2 and 4.3 summarize the results for both the activity-based and the temporal approach. In this table, P_{all}, P_{adj}, and P_{ada} refer to our activity-based approach using respectively the *all alerts in the activity*, *alerts in adjacent steps*, and *adaptive alert selection* policies discussed in Section 4.1.3. For the temporal correlator, T_{10s}, T_{1m} and T_{5m} refer respectively to window sizes of 10s, 1min and 5min. In comparison, an activity's average duration is around 5 min and can range anywhere

Table 4.2: Evaluation results : FCR and MCR

	FCR (%)						**MCR (%)**					
	Activity			**Temporal**			**Activity**			**Temporal**		
Trace	P_{all}	P_{adj}	P_{ada}	T_{10s}	T_{1m}	T_{5m}	P_{all}	P_{adj}	P_{ada}	T_{10s}	T_{1m}	T_{5m}
N°1	0	0	0	0	0	0	0	21	21	75	36	37
N°2	0	0	0	12	9	11	4	15	15	53	10	8
N°3	0	0	0	9	8	5	0	15	34	62	34	0
N°4	2	2	2	1	15	21	1	14	19	62	34	6

Table 4.3: Evaluation results : Reduction rate

	Reduction (%)					
	Activity			**Temporal**		
Trace	P_{all}	P_{adj}	P_{ada}	T_{10s}	T_{1m}	T_{5m}
N°1	84	78	78	36	61	61
N°2	82	80	80	65	70	72
N°3	84	81	79	65	70	78
N°4	83	82	81	69	77	80

between 1 and 18 min. We set the maximum temporal window size to 5min to limit false correlations on alerts belonging to multiple activities.

Overall, the results show that the activity-based approaches, in particular using policy P_{all}, achieve the best FCR and MCR scores with the highest reduction in alerts across all traces.

With regards to the FCR, since the temporal correlator links all alerts which fall within the window without taking into account the compatibility of the attributes, our approach performs better as expected.

For instance, the temporal correlator does not distinguish between alerts relative to attacks on different PLCs. We also note that, in comparison with the temporal correlator, the activity-based policies achieve better or comparable reduction results while still maintaining a low FCR. In fact, all false correlations using the activity-based approaches involve rare cases of unknown OPC-UA flow alerts that correspond to keep-alive messages.

With respect to the MCR, we observe that the activity-based approach with policy P-1 performs the best. In contrast, the temporal approaches perform badly, especially for small time windows. Even though fewer correlations are missed by the temporal approach as the temporal window size gets bigger (T-1m and T-5m), deciding on a specific time window size remains problematic. For instance, a window size of 5 minutes captures all the expected correlations in Trace n°3 but misses 37% of the expected correlations in

Trace n°1. Instead, the results show that better performance can be achieved by following an activity-based approach. The worse relative performance of the activity-based policies P_{adj} and P_{ada} compared to P_{all} indicates that activity-long attack manifestations are not exclusively limited to adjacent slots and do not appear at each slot of the activity

Table 4.4: Alerts reported by an attack manipulating valve vp3 to overflow tank TK3

Identifier	IDS	Alert
(A11)	SOAP PB	(*src IP*,10.10.3.6), (*dst IP*, 10.10.5.2), (*dst Port*,80), (*start address*,41), (*values*,1)
(A12)	SOAP PB	(*src IP*,10.10.3.6), (*dst IP*, 10.10.5.2), (*dst Port*,80), (*start address*,41), (*values*,0)
(A13)	PP	(*component*,vp4), (*event*, ↓)
(A14)	PP	(*component*,vp4), (*event*,↑)
(A15)	PP	(*component*,vp4), (*event*,↑)
(A16)	Modbus PB	(*src IP*,10.10.3.27), (*dst IP*, 10.10.5.2), (*dst Port*,502), (*function code*,5), (*address*,40),(*data*,1)
(A17)	PP	(*component*,vp3), (*event*,↑)
(A18)	PP	(*component*,vp3), (*event*,↑)
(A19)	PP	(*component*,vp3), (*event*,↑)

To illustrate this point, consider an attack which consists in overflowing tank *TK3*. The attacker manipulates valves *vp3* and *vp4* through different protocols. In this case, the attacker uses SOAP to manipulate valve *vp3* and Modbus to manipulate valve *vp4*. In normal functioning, valves *vp3* and *vp4* are only used to input a precise amount of reactants P3 and P4 into the tank *TK3*. This is performed at the first two steps of the activity. Afterward, the tank is full and valves *vp3* and *vp4* are no longer manipulated until a new cycle of the activity where the tank is empty. To overflow the tank, the attacker opens valves *vp3* and *vp4* in steps of the activity where the tank is already full.

This attack causes 6 physical process (PP) IDS alerts spread throughout the activity and 3 cyber-domain alerts from the SOAP and Modbus payload-based (PB) IDS. In response to this attack, the activity-based approach with policy P_{all} generates a meta-alert containing all of the aforementioned alerts. The details of these alerts are reported in Table 4.4. For each alert, the table reports its identifier, the source IDS and the information contained in the alert as attribute-value pairs. Figure 4.8 shows the correlation graph corresponding to the meta-alert. Nodes in the graph refer to alerts, while edges refer to the correlations. Edge labels specify the strength of the correlations (*strong* or *weak*), the attributes that are weakened in case of a weak correlation, and the absolute reception time difference between the correlated alerts in seconds.

From Figure 4.8, we see several strong and weak correlations between these alerts. Some strong correlations are straightforward. For instance, alerts *A14* and *A15* report exactly the same information regarding valve *vp4* and are thus strongly correlated. On the other hand, the cross-domain strong correlation between alerts *A11* and *A14* requires the mapping of the physical alert *A11* into the cyber domain. In this case, *A11* and *A14*

are correlated since *A14* reports the forbidden opening of valve *vp4* which is controlled by the PLC at address 10.10.5.2. This PLC can be queried using the SOAP protocol and is configured to associate *vp4* to address 41. The same reasoning holds for the strong cross-domain correlations between *A12* and *A13*, and between *A16* and *A17,A18,A19*.

Figure 4.8 shows also weak correlations both within the same domain and across domains. For instance, alerts *A11* and *A12* both report an unknown SOAP command from the same host, towards the same PLC, and writing to the same address 41. However, they differ in the *values* which are written. Thus, they are weakly correlated by ignoring the *values* attribute. We can possibly interpret this correlation as a single attacker performing different operations on the same actuator (for instance opening and closing a valve).

We can observe in Figure 4.8 that the alerts from the Modbus IDS (*A16*) and SOAP IDS (*A11*,*A12*) are never directly correlated. Yet they belong to the same meta-alert. We observe instead that both they are indirectly correlated through alerts within the physical domain. For instance, *A11* is strongly correlated with physical domain alert *A15*, *A16* is strongly correlated with physical domain alert *A19*, and physical domain alerts *A15* and *A19* are weakly correlated (by weakening the *component* attribute since they both report the opening of valves which are controlled by the same PLC). Moreover, both *A11* and *A16* are weakly correlated with the same physical domain alert *A13*. This example illustrates our discussion in Section 4.1.5 about allowing indirect correlations within meta-alert as a way to group alerts which, while not correlated in the same domain, are indirectly correlated in the other domain.

We also observe that the absolute time difference between alerts varies widely from 0s to 555s. Thus, depending on the selected time window size, a subset of the alerts is successfully correlated by the temporal approach. Note that all edges from the node A19 have large time values. In fact, A19 is recorded many steps after the initial cluster of alerts, thus alert selection policies P_{adj} and P_{ada} will also fail to capture it.

4.4 Conclusion

In this chapter, we have developed an ICS-oriented alert correlation approach to link heterogeneous alerts from IDS spanning both the cyber and physical domains. Our approach includes three main stages. First, an alert selection stage selects the alerts to be correlated based on the evolution of the activities within the physical process. Then, an alert enrichment stage carries the alerts generated by the physical domain IDS developed in Chapter 3 into the cyber domain so that they can be correlated with alerts from classical cyber domain IDS. Finally, an alert aggregation stage generates meta-alerts that contain alerts spanning both the physical and the cyber domains. The main contributions of this chapter reside in two aspects: the mapping of physical domain alerts to the cyber domain through abstraction operators and the definition of alert selection policies that take into account the runtime context of the physical process. The evaluation of our approach on a complex physical process subject to process-oriented attacks has shown good correlation metrics compared to temporal-based correlators.

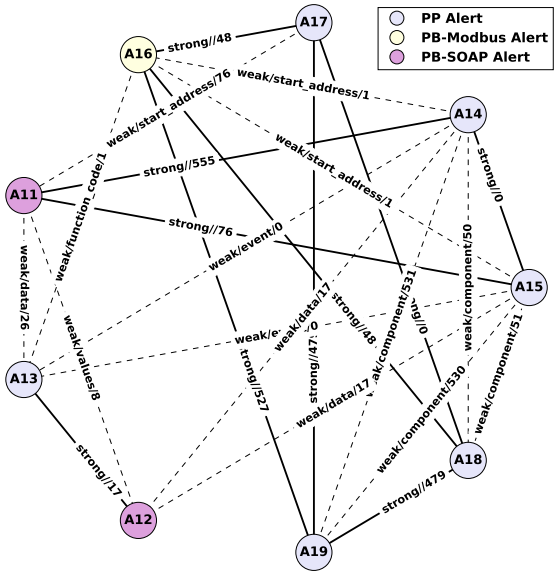


Figure 4.8: Correlation graph of a long-term attack using the activity-based approach with policy P-1

Conclusion and perspectives

In this thesis, we have presented two main contributions: (i) a physical process-oriented intrusion detection approach based on the efficient mining and monitoring of specifications over the sequential dynamics of an ICS, and (ii) an alert correlation approach which enriches and maps physical domain alerts into the cyber domain to enable cross-domain aggregation of alerts. We first summarize the context which has motivated these two contributions.

Throughout this work, one of our main concerns has been to factor in the specificities of ICSs with respect to traditional IT systems. First, ICSs have peculiar constraints in terms of resources and time constraints, especially at the lower levels of the system. This has justified the need to adopt non-intrusive and a posteriori security measures. As a result, we have identified intrusion detection as a promising solution to detect attacks against an ICS.

Next, we took into account the co-existence of both a physical and a cyber domain within an ICS. Because of this dual aspect, ICSs are exposed to process-oriented attacks which aim at generating incorrect behavior at the level of the physical process. This is exemplified in several of the recent high profile real security incidents [39, 138, 32]. In all these instances, the attacks have their starting point in the cyber domain and thus induce manifestations in both the cyber and the physical domains. With this in mind, our main worry was how to best detect such attacks in both their physical and their cyber manifestations.

Through a review of the literature, we have identified several approaches which attempt to answer this issue. One category of approaches relies solely on cyber-domain IDS to detect both the cyber and physical domain manifestations of the attacks. Unfortunately, the approaches in this category do not yield good detection results as they lack precise knowledge about the state of the physical process. On the other hand, the second category of approaches attempts to detect process-oriented attacks within the physical domain. While such approaches are promising, we have identified several challenges that question their practicality. First, some of the approaches lack models that are sufficiently expressive to capture the dynamics, both continuous and sequential, of an ICS. Unfortunately, the approaches with adequate expressivity still require expert knowledge to manually specify the models. In addition, most of the physical domain oriented approaches lack any vision about the cyber domain manifestations, and thus cannot help the operators identify the source of the attacks.

It is with this context in mind that we have argued for the following proposals which have been applied in our contributions:

a) The development and deployment of separate intrusion detection approaches for the cyber and the physical domains of an ICS. Against the cyber domain approaches which attempt to detect process oriented attacks based solely on cyber domain observations, we argued for the detection of physical domain manifestations based on the direct monitoring of sensor and actuator states.

b) To reduce the cost of building the detection models, we have opted to automatically mine process specifications from attack-free execution traces. This is an anomaly-based approach which, while holding the promise of detecting novel attacks, can also generate many false positives when the execution traces are not representative of the total behavior of the system. Thus, we have stressed the need for a careful mining in order to: (i) avoid overloading the operators with redundant and useless alerts, (ii) provide precise information about the location of the violation so that the operator can better decide whether it is a false or a true alert.

c) To link the cyber domain and physical domain manifestations of an attack, we have argued for the necessity of an alert correlation stage. This correlation stage complements the separation argued for in a) to provide the operators for a more comprehensive view of the attack. We have identified and attempted to answer some challenges that arise when developing a correlation approach in ICS. Chief among these is the heterogeneity of the attributes that are reported by the alerts from the different domains. This has motivated us to develop an alert pre-processing stage that maps the physical domain alerts into the cyber domain. We have also argued for the necessity to take into account the evolution of the physical process when selecting the alerts to correlate. This has lead us to adopt and evaluate several process-aware alert selection strategies.

The above proposals have been implemented and evaluated in a realistic test bed with a complex physical process under process oriented attacks. While the results show that the proposed approaches hold some promise, further extensions of this work can be envisaged to improve their scope and their applicability.

Perspectives

We now discuss some possible extensions to our ICS-oriented intrusion detection and alert correlation approaches.

Intrusion detection

- In Section 3.1, as part of our threat model, we have elected to focus on the sequential aspect of the ICS. This has motivated our choice of process specification formalisms enabling the representation of qualitative (LTL) and quantitative

(MTL) constraints over the evolution of actuator and sensor states and events. One straightforward extension of our work is to consider a broader threat model that encompasses also the continuous aspect of the ICS. This would require the adoption of an adequate formalism, such as Signal Temporal Logic (STL) [91]. STL extends LTL by allowing for predicates over real-valued variables while retaining the capability to express quantitative time constraints as in MTL. Thus, STL allows for runtime verification to cover the behaviors induced by hybrid cyber-physical systems. While STL has been so far applied in various domains such as systems biology and analog circuits, applications to intrusion detection remain to be explored. In keeping with the general spirit of our approach, a first step would be to explore and collect useful specification patterns based on STL.

- As discussed in the analysis our intrusion detection approach's evaluation in Section 3.4, in addition to the binary information concerning the violation of the quantitative property, an operator might also be interested in a quantitative estimation of the deviation with respect to the constraint. For instance, the operator might be interested in knowing that, for a valve that should be opened for at most every $30s$, it was, in fact, opened every $2s$ as a result of a wear attack on the valve. Recent research within runtime verification has begun exploring such issues by developing such monitoring techniques for certain classes of metric temporal logic properties [31].

- Using feedback by the security operator on the pertinence of the process specifications, we might be able to further adjust their level of precision. Recall from Section 3.2 that we aim for maximal precision of the specification scopes, but that this can induce an increase in the number of mined specifications. In the case of constraints that apply on contiguous scopes, our approach would mine a specification for each scope. However, an operator might require this level of precision, and would prefer to have a single specification spanning all these scopes while retaining the possibility to drill down whenever more precision is necessary.

- In this thesis, we have assumed that each monitor has access to all the sensor and actuator states it requires to decide whether a property is violated or not. However, this need not be the case in general. For instance, properties might be specified over sensors and actuators which are distributed over multiple PLCs. In this case, multiple monitors might be dispatched, each one having access only to a subset of the actuator and sensor states which it needs to give a verdict. To correctly monitor such global properties, some form of cooperation between the distributed monitors is necessary [11] where information about the violation of the property is exchanged between the monitors. However, exchanging information involves latencies which can lead to a delay in the emission of the verdict. As a result, the attacks might be detected too late for the operators to properly react. Research is still needed to decide on the best way to distribute the monitors so as to minimize such delays for the purpose of intrusion detection.

Alert correlation

- Recall that our approach is primarily concerned with aggregating activity-long alerts related to the elementary steps of an attack. Thus, a natural extension consists in reconstructing complex attack scenarios spanning multiple activities. Such reconstruction could rely on a joint security-safety analysis [73] to produce the attack scenarios, and on the activity-specific meta-alerts generated by our approach to track the elementary attack steps.

- In the case of online detection, the time window for an operator to diagnose the situation and react accordingly can be limited. Thus, one possible extension of this work concerns the development of safe intrusion prevention systems (IPS) that can react automatically to abnormal events and afford more time for an operator's intervention. The development of an IPS for ICSs is challenging because of the high cost involved in case of a wrong decision based on false alerts. Ultimately, better decisions can be expected if information from both the cyber and the physical domains are included in the decision process. An interesting line of research to follow would be to use the information gathered within the clusters generated by our correlation approach to guide the actions performed by an IPS. For instance, an IPS would react differently if a cluster relates a process violation with an unknown flow coming from a machine that is not supposed to interact with the physical process, in comparison with a process violation with an unknown actuator manipulation command coming from a legitimate supervisor. While the former can be readily interpreted as an attack, the second is less clear and might need careful actions as to not block a legitimate supervisor from performing its duties.

Acronyms

AMI Advanced Metering Infrastructure.

C&C Command & Control.

DCS Distributed Control System.

DNF Disjunctive Normal Form.

DNP Distributed Network Protocol.

DPA Deep Protocol Analysis.

FBD Function Block Diagram.

FPR False Positive Rate.

HIDS Host Intrusion Detection System.

HTTP HyperText Transport Protocol.

IACS Industrial Automation and Control System.

ICS Industrial Control System.

IDMEF Intrusion Detection Message Exchange Format.

IDS Intrusion Detection System.

IEC International Electrotechnical Commission.

IED Intelligent Electronic Device.

IL Instruction List.

IP Internet Protocol.

IT Information Technology.

LTL Linear Temporal Logic.

MTL Metric Temporal Logic.

NIDS Network Intrusion Detection System.

OPC DA Open Platfrom Communication Data Access.

OPC UA Open Platform Communication Unified Architecture.

OS Operating System.

PCS Process Control System.

PID Proportional Integral Derivative.

PLC Programmable Logical Controller.

RPC Remote Procedure Call.

RTU Remote Terminal Unit.

SCADA Supervisory Control and Data Acquisition.

SFC Sequential Function Chart.

SIEM Security Information and Event Management.

SMT Satisfiability Modulot Theories.

SOAP Simple Object Access Protocol.

ST Structured Text.

TCP Transport Control Protocol.

TPR True Positive Rate.

UDP User Datagram Protocol.

WCET Worst Case Execution Time.

Appendix A

Basics of Sequential Function Charts (SFC)

Sequential Function Chart (SFC) is a graphical language, defined in IEC61131-3 [64], particularly adapted to the specification of the sequential control requirements of a system. An SFC represents sequential tasks as a combination of steps, actions, and transitions as represented in Figure A.1.

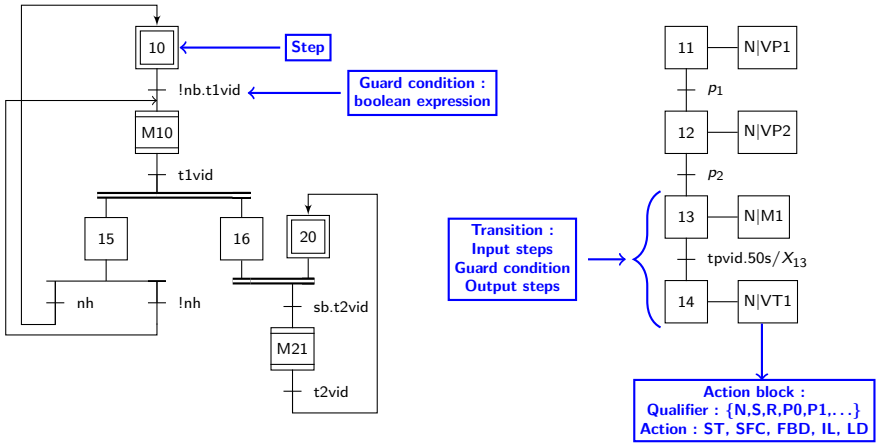


Figure A.1: Example of an SFC with its main constituents

Steps Steps are depicted as rectangular boxes and represent particular states of the controlled system. Each SFC defines at least one initial step (steps 10 and 20 in Figure A.1) where control must start during execution. When control reaches a step during execution, we say that the step is active. Each step is associated with action blocks which define the actions to be performed when the step becomes active. During an execution cycle, only the actions associated with the active steps are evaluated.

Transitions Steps are connected using transitions which are depicted as horizontal lines between two steps. Transitions link a set of input steps with a set of output steps through condition guards which are boolean expressions over input and internal variables. When the input steps of a transition are active, and the guard condition is satisfied, the process leaves the input steps and the output steps become active. During an execution cycle, only the transitions following active steps are evaluated.

Actions The actions that are carried out in steps are depicted using action blocks. An action block consists of two main parts: an action qualifier and an action. The action provides a description of the action to be performed, possibly in one of the five programming languages defined in IEC 61131-3 [64]. The action qualifier specifies when the action must be executed. For example, an action might be executed once when the step is activated (qualifier P), throughout the activation of the step (qualifier N), or with some time delay after the step is activated (qualifier D). A complete description of all possible qualifiers can be found in [64].

Control structures Steps and transitions within an SFC can be organized into six main control structures.

Linear sequence Represents a linear succession of steps and transitions. This is represented in Figure A.1, by the sequence of steps 11 to 14. In this control structure, when control leaves an active step, it can move only to the next step in the sequence.

Divergent selection sequences This control structure links one input steps with multiple output steps where each output step is preceded by a transition. This situation is depicted in Figure A.2. Here, execution can move from step 1 to either step 2 or step 3 depending on the evaluations of transitions t1 and t2. When multiple transitions are satisfied, an order, typically from left to right, is applied to decide which transition to take.

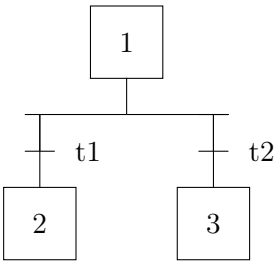


Figure A.2: Divergent selection sequence in an SFC

Convergent selection sequences This control structure merges multiple branches resulting from a divergent selection sequence into a single branch. In this case, transitions from each of the branches are followed by a single step as depicted in Figure A.3.

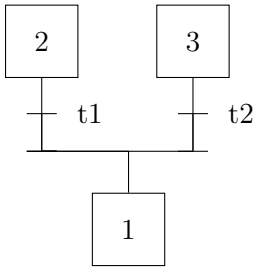


Figure A.3: Convergent selection sequence in an SFC

Divergent parallel sequences Contrary to the divergent selection sequence case, divergent parallel sequences allow for multiple branches to hold active steps simultaneously. This is performed by following a single transition with multiple output steps. For instance, in Figure A.4, when step 1 is active and transition evaluates to true, both steps 1 and 2 become simultaneously active.

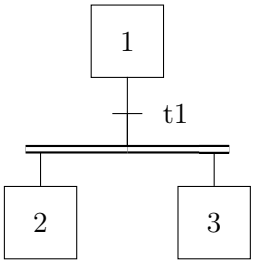


Figure A.4: Divergent parallel sequence in an SFC

Convergent parallel sequences A convergent parallel sequence ends a divergent parallel sequence. In Figure A.5, steps 2 and 3 are the last steps in two separate parallel branches. When both steps become active, and the transition t1 is satisfied, then control moves to step 1.

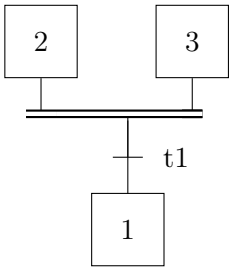


Figure A.5: Convergent parallel sequence in an SFC

Appendix B

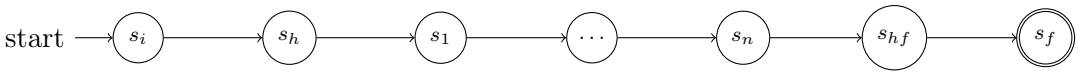
Process-oriented attack parameters

The attacks are classified depending on their type (I or II, see Section 3.3). For each entry, we specify the duration parameters associated with each state (in seconds) and the action performed when execution reaches a state. Every action specifies the target PLC, the target actuator and its new assigned value. For instance, the line :

$$s_1(1000, 200), \text{PLC@10.10.5.1}, \text{Actuator@}VP1, \text{Value} = 1$$

means that the duration parameters for state s_1 are $(\mu = 1000, \delta = 200)$, the target PLC is at address 10.10.5.1, the target variable is $VP1$ and its new value is 1. Note also that for the correlation approach, each action might be performed by a different protocol, the choice of which is decided randomly at runtime (see Section 4.2).

Type I attacks




```
si(2000,500)
sh(2000,500), PLC@10.10.5.1, Actuator@Manual, Value=1
shf(2000,500), PLC@10.10.5.1, Actuator@Manual, Value=0
s1(20000,2000), PLC@10.10.5.1, Actuator@VP2, Value=1
s2(2000,500), PLC@10.10.5.1, Actuator@VP2, Value=0
sf(2000,500)
```

Listing B.1: Inserting a bad balance of products P1 and P2 in TK1

```
si(2000,500)
sh(2000,500), PLC@10.10.5.1, Actuator@Manual, Value=1
```

```
shf(2000,500), PLC@10.10.5.1, Actuator@Manual, Value=0
s1(20000,500), PLC@10.10.5.1, Actuator@VP2, Value=1
s2(20000,2000), PLC@10.10.5.1, Actuator@VP1, Value=1
s3(2000,500), PLC@10.10.5.1, Actuator@VP2, Value=0
s4(2000,500), PLC@10.10.5.1, Actuator@VP1, Value=0
sf(2000,500)
```

Listing B.2: Inserting a bad balance of products P1 and P2 in TK1

```
si(2000,500)
sh(2000,500), PLC@10.10.5.1, Actuator@Manual, Value=1
shf(2000,500), PLC@10.10.5.1, Actuator@Manual, Value=0
s1(20000,2000), PLC@10.10.5.1, Actuator@VT2, Value=1
s2(2000,500), PLC@10.10.5.1, Actuator@VT2, Value=0
sf(2000,500)
```

Listing B.3: Introducing unfinished reactant in S1

```
si(2000,500)
sh(2000,500), PLC@10.10.5.1, Actuator@Manual, Value=1
shf(2000,500), PLC@10.10.5.1, Actuator@Manual, Value=0
s1(2000,2000), PLC@10.10.5.1, Actuator@M2, Value=1
s2(2000,500), PLC@10.10.5.1, Actuator@M2, Value=0
sf(2000,500)
```

Listing B.4: Stopping mixing of reactant in TK2 short of 60s

```
si(2000,500)
sh(2000,500), PLC@10.10.5.2, Actuator@Manual, Value=1
shf(2000,500), PLC@10.10.5.2, Actuator@Manual, Value=0
s1(20000,2000), PLC@10.10.5.2, Actuator@VP4, Value=1
s2(2000,500), PLC@10.10.5.2, Actuator@VP4, Value=0
sf(2000,500)
```

Listing B.5: Inserting a bad balance of products P3 and P4 in TK2

```
si(2000,500)
sh(2000,500), PLC@10.10.5.2, Actuator@Manual, Value=1
shf(2000,500), PLC@10.10.5.2, Actuator@Manual, Value=0
s1(2000,2000), PLC@10.10.5.2, Actuator@M4, Value=1
```

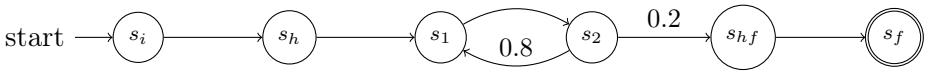
```
s2(2000,500), PLC@10.10.5.2, Actuator@M4, Value=0
sf(2000,500)
```

Listing B.6: Stopping mixing of reactant in TK4 short of 60s

```
si(2000,500)
sh(2000,500), PLC@10.10.5.2, Actuator@Manual, Value=1
shf(2000,500), PLC@10.10.5.2, Actuator@Manual, Value=0
s1(20000,2000), PLC@10.10.5.2, Actuator@VT4, Value=1
s2(2000,500), PLC@10.10.5.2, Actuator@VT4, Value=0
sf(2000,500)
```

Listing B.7: Introducing bad quality product in S2

Type II attacks




```
si(2000,500)
sh(2000,500), PLC@10.10.5.1, Actuator@Manual, Value=1
shf(2000,500), PLC@10.10.5.1, Actuator@Manual, Value=0
s1(500,100), PLC@10.10.5.1, Actuator@M1, Value=1
s2(500,100), PLC@10.10.5.1, Actuator@M1, Value=0
sf(2000,500)
```

Listing B.8: Wearing motor M1 through quick start/stop commands

```
si(2000,500)
sh(2000,500), PLC@10.10.5.2, Actuator@Manual, Value=1
shf(2000,500), PLC@10.10.5.2, Actuator@Manual, Value=0
s1(500,100), PLC@10.10.5.2, Actuator@M3, Value=1
s2(500,100), PLC@10.10.5.2, Actuator@M3, Value=0
sf(2000,500)
```

Listing B.9: Wearing motor M3 through quick start/stop commands

```
si(2000,500)
sh(2000,500), PLC@10.10.8.100, Actuator@Manual, Value=1
shf(2000,500), PLC@10.10.8.100, Actuator@Manual, Value=0
s1(500,100), PLC@10.10.8.100, Actuator@R1On, Value=1
s2(500,100), PLC@10.10.8.100, Actuator@R1On, Value=0
sf(2000,500)
```

Listing B.10: Tampering with the reactor during the chemical reaction

Appendix C

Description of some common ICS protocols

Modbus/TCP. Modbus/TCP protocol [135] is an application layer protocol operating in a client/server setting. It is typically used for inter-PLC and supervisor/HMI to PLC communications. The structure of a Modbus/TCP message is given in Figure C.1. In particular, the *transaction ID* field allows to uniquely match a response to its corresponding request, the *function code* specifies the nature of the request (for instance read discrete inputs (1 bit) or registers (16 bits)) while the *data* field depends on the function code (in case of a read request, the data field can specify the start address and number of elements to read). The length of the data field varies and can be computed using the length field. The *protocol ID* field is used for intra-system multiplexing and takes the value of 0 for the Modbus/TCP protocol. The *unit id* field is used for intra-system routing purposes, for example, to reach a Modbus slave device in a Modbus serial line through a gateway between a TCP/IP network and a Modbus serial line.

DNP3. DNP3 [61] is a non-proprietary SCADA protocol designed for communications between master stations and remote substation including RTUs and IEDs. While DNP3 is primarily used in the electric utility industry, it has increasingly been adopted in adjacent industries such as water treatment, oil and gas, and transportation. DNP3 application layer messages are broken into fragments whose maximum size lies between 2048 to 4096 bytes depending on the receiving device's buffer size. As depicted in Figure C.2, a DNP3 application layer message contains an *application header* and one or more *DNP3 objects*, each data object is associated with an *object header*.

Transaction ID	Protocol ID	Length	Unit ID	Function Code	Data
2 Bytes	2 Bytes	2 Bytes	1 Byte	1 Byte	n Bytes

Figure C.1: Structure of a Modbus/TCP message

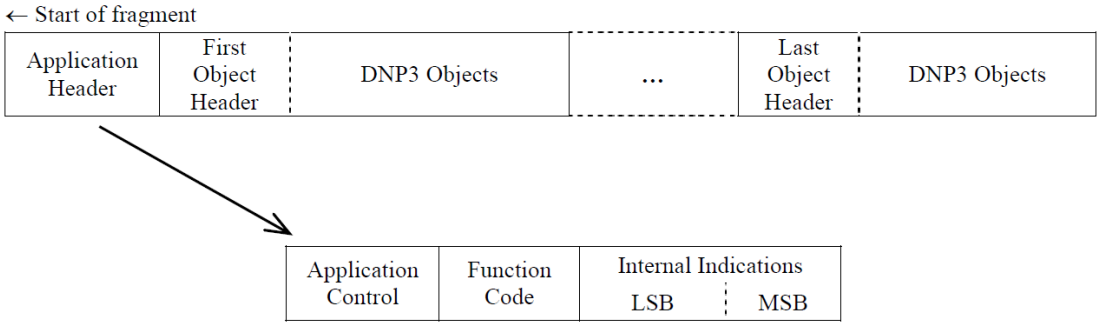


Figure C.2: Structure of a DNP3 fragment and DNP3 application header [50]

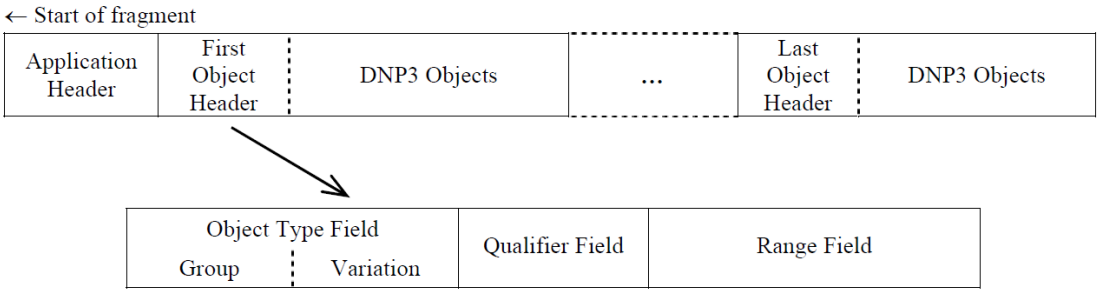


Figure C.3: Structure of a DNP3 object header [50]

The application header differs depending on whether the message corresponds to a request or a response. In case of a request, the application header includes an *application control* field and a *function code* field. The 1-byte application control field is used to reassemble multiple fragments, detect duplicate fragments, and manage acknowledgments. The function code field defines the purpose of the fragment and specifies for instance whether a request is to read (function code 0x1) or write (function code 0x2) data. In addition to the application control and function code fields, application headers in response fragments also include a 2-bytes *internal indications* field that specifies the status of the responding device (need time, device restart, etc.), and information about the request (function code not supported, object unknown, etc.).

DNP3 organizes data (binary input/output, analog input/output, counters, etc.) in terms of objects. When querying or returning objects, DNP3 devices use *object headers* (see Figure C.3) within DNP3 fragments to specify the expected type of objects and the range of objects to be queried/returned. Within an object header, the *object type field* indicates the *group* or types of the object (for instance, group 10 for binary outputs or group 40 for analog outputs), and the *variation* or format (for instance, analog inputs can be formatted using 16 or 32 bit integers, or using a floating-point representation). Object headers also include a *qualifier field* and a *range field* which together indicate the

object addressing method. Objects can be addressed either by specifying a *start* and *end* object index, by specifying a *start* index and a *count* of the number of objects addressed, or by addressing *all* objects of a given type (no range specified). Finally, alongside object headers, *data objects* contain the data values to be written or to be returned depending on the type of the DNP3 fragment as specified by the headers (function code, group, variation, etc.).

IEC-104. IEC 60870-5-104 (also known as IEC-104) [136] is a SCADA protocol defined within the IEC 60870 collection of standards. IEC-104 is the TCP/IP adaptation of the older IEC 60870-5-101 (IEC-101) serial protocol. The IEC-101 defines the functionalities necessary for remote control over wide areas. Both IEC-104 and IEC-101 are used extensively within the electrical industry, for instance, to establish communications between electrical control stations and substations. In this chapter, we focus on the TCP/IP based IEC-104 protocol.

In its application layer, IEC-104 defines two sub-layers: the Application Protocol Control Information (*APCI*) sub-layer, and the Application Service Data Unit (*ASDU*) sub-layer. The APCI sub-layer, which lies on top of the TCP layer, defines three types of IEC-104 message formats: information transfer format (*I-format*), numbered supervisory functions (*S-format*), and unnumbered control functions (*U-format*). U-format messages are used to start, stop or check the status of connections. I-format messages are used to transfer data between IEC-104 devices. S-format messages are used to acknowledge previously received I-format messages. Since we are interested in data transfer between devices, we focus on I-format messages. Note that only I-format messages carry ASDU sub-layer payloads, while S-format and U-format messages have a fixed length and only carry ACPI.

An IEC-104 ASDU, whose structure is depicted in Figure C.4, consists of a *data unit identifier* along with one or more *information objects*. The data unit identifier specifies the type of the accessed data through the *type identification* field (for instance 45 for commands to access binary outputs). In IEC-104, data is organized as *information objects*, each object is characterized by one or more *information elements*. The IEC-104 standard defines the structure of the information elements that form each type of information object. Data can be addressed either by querying a sequence of information objects (including all their information elements) or by querying a sequence of information elements within a single information object. The choice between these two types of addressing modes is specified by the *structure qualifier (SQ)* field and the number of information objects or elements to be queried in the sequence is determined by the *number of objects* field.

Besides the above fields, the data unit identifier carries a *Test (T)* field which indicates whether the ASDU was generated in test conditions and is not intended to control the process, a *positive/negative (P/N)* field which provides information about the correct execution or not of a command, and a *cause of transmission (COT)* field which is used to direct the ASDU to the correct task within a station and assist in an ASDU's interpretation. Examples of COT include *background interrogation*, *data transmission*, and

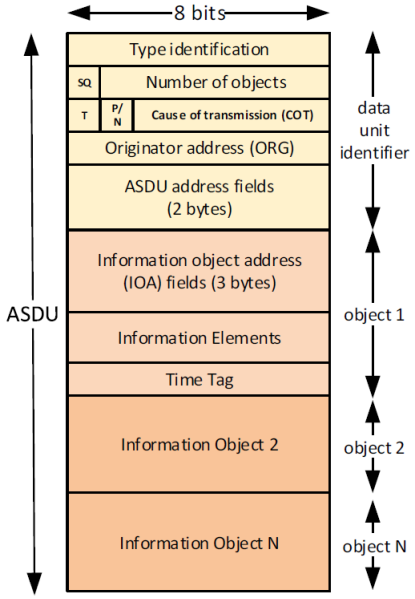


Figure C.4: Structure of an IEC-104 ASDU [92]

Information object address unknown. The data unit identifier also specifies the *originator address* which identifies the controlling station in the case of multiple controlling stations, and the *ASDU address field* which identifies either a specific target station address or a broadcast address.

Following the data unit identifier comes a number of information objects, each one characterized by an *information object address (IOA)*, one or more information elements and an optional time tag. The number and type of information objects or elements are determined by the fields in the data unit identifier.

Ethernet/IP. Ethernet/IP [108] is an industrial protocol, maintained by Open Device Vendor Association (ODVA), which adapts the Common Industrial Protocol (CIP) [107] to Ethernet-based TCP/IP networks. CIP specifies messages and services such as real-time control, time synchronization, energy optimization, and network management to enable communication within industrial applications. Ethernet/IP supports two types of communications: request-response based communications which are called *explicit messaging*, and pre-established connections where known information is exchanged at fixed time intervals and which is called *implicit messaging*. Implicit messaging does not require the use of address and service information since consuming nodes already know (i.e *implicitly*) the expected data based on the connection ID.

Information within each CIP node in the system is modeled as a collection of objects which are structured into classes, instances, and attributes following a classical object-oriented approach. Objects within the system can be addressed using a uniform

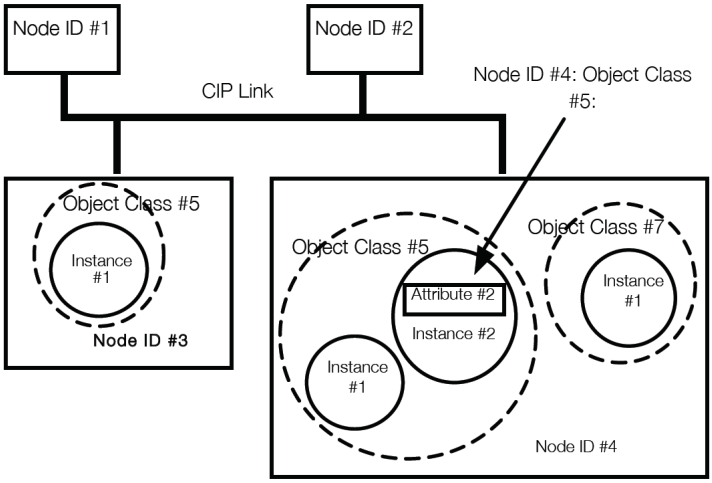


Figure C.5: Addressing scheme of CIP used by Ethernet/IP [106]

addressing scheme consisting of C.5: a *node address*, a *class identifier*, an *instance identifier*, an *attribute identifier*, and a *service code*. Each node on a CIP network is assigned a *node address*, which in the case of Ethernet/IP is represented by an IP address. Class, instance and attribute identifiers are integer identification values which, together, allow for attribute-level access to object instances. These identifiers can either be publicly defined or vendor-specific. CIP defines a large collection of commonly defined objects, including for instance discrete output point and discrete input point objects. Service codes are integer identification values which refer to an action request directed at an object instance or attribute. Examples of commonly used services include read and write functions. As in the case of objects, CIP defines a library of common services and allows for vendor-specific services to be defined. To specify the set of objects and services provided by a device, simple ASCII-based text files called Electronic Data Sheets (EDS) can be used.

Figure C.6 depicts the structure of a CIP request message. The *Request service* field indicates the type of action performed on the object referenced by the request. The *Request Path Size* field specifies the number of 16-bit words in the *Request Path*. The Request Path contains one or more references to class, instance, and attribute identifiers. References can either be in the form of 8-bit, 16-bit or 32-bit identifiers, or in the form of a symbolic name using ANSI Extended Symbol Segments defined by the CIP specifications. Finally, the *Requested Data* field contains service-specific data provided to the object referenced in the Requested Path field.

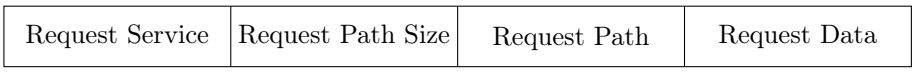


Figure C.6: Structure of a CIP request message

Bibliography

[1] S. Adepu and A. Mathur. Using process invariants to detect cyber attacks on a water treatment system. In *Proc. 31st IFIP TC 11 Conf.*, pages 91–104. Springer, 2016. 49

[2] Maryam Raiyat Aliabadi, Amita Ajith Kamath, Julien Gascon-Samson, and Karthik Pattabiraman. Artinali: Dynamic invariant detection for cyber-physical system security. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2017, pages 349–361, New York, NY, USA, 2017. ACM. 24

[3] Argus. *http://qosient.com/argus/*. [Online, acc. March-2018]. 100

[4] Eugene Asarin, Alexandre Donzé, Oded Maler, and Dejan Nickovic. Parametric identification of temporal properties. In Sarfraz Khurshid and Koushik Sen, editors, *Runtime Verification*, pages 147–160, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. 64

[5] R. Barbosa, R. Sadre, and A. Pras. Flow whitelisting in SCADA networks. *Int. Journal of Critical Infrastructure Protection*, 6(3-4):150–158, December 2013. 20, 21, 100

[6] Rafael Ramos Regis Barbosa, Ramin Sadre, and Aiko Pras. Difficulties in Modeling SCADA Traffic : A Comparative Analysis. In *Proceedings of the 13th international conference on Passive and Active Measurement (PAM '12)*, pages 126–135, 2012. ix, 20, 21

[7] Ezio Bartocci, Yliès Falcone, Adrian Francalanza, and Giles Reger. *Introduction to Runtime Verification*, pages 1–33. Springer International Publishing, Cham, 2018. 29, 30, 32

[8] David Basin, Bhargav Nagaraja Bhatt, and Dmitriy Traytel. Almost event-rate independent monitoring of metric temporal logic. In Axel Legay and Tiziana Margaria, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 94–112, Berlin, Heidelberg, 2017. Springer Berlin Heidelberg. 33

[9] David Basin, Felix Klaedtke, and Eugen Zălinescu. Algorithms for monitoring real-time properties. *Acta Informatica*, 55(4):309–338, Jun 2018. 31

[10] Andreas Bauer. Monitorability of omega-regular languages. *CoRR*, abs/1006.3638, 2010. 32

[11] Andreas Bauer and Yliès Falcone. Decentralised ltl monitoring. *Form. Methods Syst. Des.*, 48(1-2):46–93, April 2016. 111

[12] Andreas Bauer, Martin Leucker, and Christian Schallhart. Runtime verification for ltl and tltl. *ACM Trans. Softw. Eng. Methodol.*, 20(4):14:1–14:64, September 2011. 33

[13] Carlo Bellettini and Julian L. Rrushi. A product machine model for anomaly detection of interposition attacks on cyber-physical systems. *IFIP International Federation for Information Processing*, 278:285–299, 2008. 21

[14] R. Berthier, W.H. Sanders, and H. Khurana. Intrusion Detection for Advanced Metering Infrastructures: Requirements and Architectural Directions. *Smart Grid Communications (SmartGridComm), 2010 First IEEE International Conference on*, 2010. 25, 26

[15] Robin Berthier and William H. Sanders. Specification-based intrusion detection for advanced metering infrastructures. In *Proceedings of the 17th IEEE Pacific Rim International Symposium on Dependable Computing, PRDC*, pages 184–193, 2011. 25, 26

[16] Yves Boussemart and Mary L. Cummings. Predictive models of human supervisory control behavioral patterns using hidden semi-markov models. *Engineering Applications of Artificial Intelligence*, 24(7):1252 – 1262, 2011. Infrastructures and Tools for Multiagent Systems. 68

[17] L. Briesemeister, S. Cheung, U. Lindqvist, et al. Detection, correlation, and visualization of attacks against critical infrastructure systems. In *8th Annual Conf. on Privacy, Security and Trust*, Ottawa, Canada, 2010. 26, 39

[18] Bro. `http://www.bro.org/`. [Online, acc. March-2018]. 100

[19] a. Carcano, a. Coletta, M. Guglielmi, M. Masera, I. Nai Fovino, and a. Trombetta. A multidimensional critical state analysis for detecting intrusions in SCADA systems. *IEEE Transactions on Industrial Informatics*, 7(2):179–186, 2011. ix, 16, 23, 25, 26, 41

[20] Andrea Carcano, Igor Nai Fovino, Marcelo Masera, and Alberto Trombetta. State-based network intrusion detection systems for SCADA protocols: A proof of concept. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6027 LNCS:138–150, 2010. 22, 23, 24, 25

[21] A. Cárdenas, A. Saurabh, et al. Challenges for securing cyber physical systems. In *Workshop on Future Directions in CPS Security*, July 2009. 7, 8, 17

[22] Alvaro Cárdenas, Saurabh Amin, Zong-Syun Lin, Yu-lun Huang, Chi-Yen Huang, and Shankar Sastry. Attacks Against Process Control Systems: Risk Assessment, Detection, and Response. *ACM Symposium on Information, Computer and Communications Security*, pages 355–366, 2011. 24, 49

[23] M. Caselli, E. Zambon, and F. Kargl. Sequence-aware Intrusion Detection in Industrial Control Systems. In *Proc. 1st ACM Workshop CPSS*, pages 13–24, 2015. ix, 18, 19, 22, 25, 41, 67

[24] S. Cheung and K. Skinner. Using Model-based Intrusion Detection for SCADA Networks. In *Proc. SCADA Security Scientific Symposium*, pages 127–134, 2007. 18, 20

[25] E.M. Clarke, D. Peled, and O. Grumberg. *Model Checking*. MIT Press, 1999. 27

[26] CVC4. *http://cvc4.cs.stanford.edu/*. [Online, acc. March-2018]. 100

[27] Marcelo D'Amorim and Grigore Roşu. Efficient Monitoring of ω-languages. In *Proc. CAV'05*, pages 364–378, 2005. 33

[28] Agence Nationale de la Sécurité des Systèmes d'Information (ANSSI). *Maîtriser la SSI pour les systèmes de contrôle industriels*. ANSSI, June 2012. 2

[29] H. Debar and A. Wespi. Aggregation and correlation of intrusion-detection alerts. In *RAID*, pages 85–103, Berlin, 2001. Springer. 26, 37, 38, 77

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

[31] Adel Dokhanchi, Bardh Hoxha, and Georgios Fainekos. On-line monitoring for temporal logic robustness. In Borzoo Bonakdarpour and Scott A. Smolka, editors, *Runtime Verification*, pages 231–246, Cham, 2014. Springer International Publishing. 76, 111

[32] Dragos. Crashoverride: Analysis of the threat to electric grid operations. lhttps://dragos.com/blog/crashoverride/, 2017. [Online, acc. : July-2018]. xiii, 12, 36, 50, 67, 69, 109

[33] Patrick Düssel, Christian Gehl, Pavel Laskov, Jens-Uwe Buß er, Christof Störmann, and Jan Kästner. *Critical Information Infrastructures Security: 4th International Workshop, CRITIS 2009, Bonn, Germany, September 30 - October 2, 2009. Revised Papers*, chapter Cyber-Critical Infrastructure Protection Using Real-Time Payload-Based Anomaly Detection, pages 85–97. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. 18

[34] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Patterns in property specifications for finite-state verification. In *Proc. ICSE'99*, 1999. 34, 35

[35] 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. 22, 23, 25

[36] Kelvin Erickson. Programmable logic controllers. *Potentials, IEEE*, 15(1):14–17, Feb 1996. 4

[37] F-Secure. Backdoor:w32/havex. https://www.f-secure.com/weblog/archives/00002718.html, 2014. [Online, acc. : March-2018]. 10

[38] Yliès Falcone, Jean-Claude Fernandez, and Laurent Mounier. What can you verify and enforce at runtime? *International Journal on Software Tools for Technology Transfer*, 14(3):349–382, Jun 2012. 32, 33

[39] N Falliere, L. O. Murchu, and E. Chien. W32.stuxnet dossier. https://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf, 2011. [Online, acc. : March-2018]. xiii, 11, 25, 109

[40] 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 *Proceedings of the 2017 Workshop on Cyber-Physical Systems Security and PrivaCy*, CPS '17, pages 13–24, New York, NY, USA, 2017. ACM. 22, 24

[41] Std for Software Verification and Validation Working Group. *IEEE Std 1012-2016 (Revision of IEEE Std 1012-2012 - IEEE Standard for System and Software Verification and Validation)*. IEEE, 2017. 27

[42] I. N. Fovino, A. Carcano, T. D. L. Murel, A. Trombetta, and M. Masera. Modbus/dnp3 state-based intrusion detection system. In *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, pages 729–736, April 2010. 22, 23, 24

[43] Brendan Galloway and Gerhard P. Hancke. Introduction to Industrial Control Networks. *IEEE Communications Surveys & Tutorials*, 2013. 2, 5

[44] Wei Gao, Thomas Morris, Bradley Reaves, and Drew Richey. On SCADA control system command and response injection and intrusion detection. *eCrime Researchers Summit eCrime 2010*, pages 1–9, 2010. 25

[45] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, and E. Vázquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers & Security*, 28(1):18 – 28, 2009. 8

[46] Chiara Ghidini, Chiara Ghidini, Fausto Giunchiglia, and Fausto Giunchiglia. A semantics for abstraction. *Artificial Intelligence*, 57, 2003. 40

[47] Erwan Godefroy. *Definition and assessment of a mechanism for the generation of environment specific correlation rules*. Theses, Supélec, September 2016. 37, 38

[48] 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. 18, 19, 22, 25, 41

[49] B. Green, M. Krotofil, and A. Abbasi. On the significance of process comprehension for conducting targeted ics attacks. In *CPS '17 Proceedings of the 2017 Workshop on Cyber-Physical Systems Security and PrivaCy, 3 November 2017, Dallas, Texas*, United States, 11 2017. Association for Computing Machinery, Inc. 94

[50] DNP3 Users Group. Dnp3 specifiation - volume 1 : Dnp3 introduction, 2002. x, X

[51] Hadeli Hadeli, Ragnar Schierholz, Markus Braendle, and Cristian Tuduce. Generating configuration for missing traffic detector and security measures in industrial control systems based on the system description files. *IEEE Conference on Technologies for Homeland Security, HST 2009*, pages 503–510, 2009. 20

[52] Hadeli Hadeli, Ragnar Schierholz, Markus Braendle, and Cristian Tuduce. Leveraging determinism in industrial control systems for advanced anomaly detection and reliable security configuration. In *IEEE Conference on Emerging Technologies and Factory Automation ETFA 2009*, pages 1–8, 2009. 20

[53] Dina Hadžiosmanović, Damiano Bolzoni, and Pieter Hartel. A log mining approach for process monitoring in scada. *Int. J. Inf. Secur.*, 11(4):231–251, August 2012. 24

[54] Dina Hadžiosmanović, Lorenzo Simionato, Damiano Bolzoni, Emmanuele Zambon, and Sandro Etalle. N-gram against the machine: On the feasibility of the N-gram network analysis for binary protocols. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7462 LNCS:354–373, 2012. 18

[55] D. Hadziosmanovic, R. Sommer, and E. Zambon. Through the Eye of the PLC: Towards Semantic Security Monitoring for Industrial Control Systems. In *Proc. ACSAC 14*, 2014. 22, 23, 25, 49

[56] Song Han, Miao Xie, Hsiao-Hwa Chen, and Yun Ling. Intrusion detection in cyber-physical systems: Techniques and challenges. *Systems Journal, IEEE*, 8(4):1049–1059, Dec 2014. 16

[57] Klaus Havelund and Grigore Rosu. Monitoring programs using rewriting. In *Proceedings of the 16th IEEE International Conference on Automated Software Engineering*, ASE '01, pages 135–, Washington, DC, USA, 2001. IEEE Computer Society. 33

[58] Klaus Havelund and Grigore Rosu. Testing linear temporal logic formulae on finite execution traces. Technical report, 2001. 33

[59] Hsi-Ming Ho, Joël Ouaknine, and James Worrell. Online monitoring of metric temporal logic. In Borzoo Bonakdarpour and Scott A. Smolka, editors, *Runtime Verification*, pages 178–192, Cham, 2014. Springer International Publishing. 33

[60] IANA. *http://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml*. [Online, acc. March-2018]. 87

[61] IEEE. Ieee standard for electric power systems communications-distributed network protocol (dnp3). *IEEE Std 1815-2012 (Revision of IEEE Std 1815-2010)*, pages 1–821, Oct 2012. IX

[62] X. Jin, A. Donzé, J. V. Deshmukh, and S. A. Seshia. Mining requirements from closed-loop control models. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(11):1704–1717, Nov 2015. 36, 64

[63] Klaus Julisch. Clustering intrusion detection alarms to support root cause analysis. *ACM Trans. Inf. Syst. Secur.*, 6(4):443–471, November 2003. 38, 97

[64] J. Karl Heinz and M. Tiegelkamp. *IEC 61131-3: Programming Industrial Automation*. Springer, 2nd edition, 2010. 4, 51, I, II

[65] R. Khan, P. Maynard, K. McLaughlin, et al. Threat analysis of blackenergy malware for synchrophasor based real-time control and monitoring in smart grid. In *ICS-CSR '16*, pages 1–11, Swindon, UK, 2016. BCS Learning & Development. 10

[66] Dongyoung Kim, HyoChan Bang, and Jung-Chan Na. Intrusion alert normalization method using awk scripts and attack name database. In *The 7th International Conference on Advanced Communication Technology, 2005, ICACT 2005.*, volume 1, pages 608–611 Vol. 1, Feb 2005. 37

[67] S.C. Kleene. *Mathematical Logic*. Dover books on mathematics. Dover Publications, 2002. 87

[68] Amit Kleinmann and Avishai Wool. Accurate modeling of the siemens s7 scada protocol for intrusion detection and digital forensics. *Journal of Digital Forensics, Security and Law*, 9(2), 2014. 18, 19

[69] Amit Kleinmann and Avishai Wool. A Statechart-Based Anomaly Detection Model for Multi-Threaded SCADA Systems. In *The 10th International Conference on Critical Information Infrastructures Security (CRITIS 15)*, 2015. 18, 19

[70] 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 '16, pages 1–12, New York, NY, USA, 2016. ACM. 18

[71] Sascha Konrad and Betty H. C. Cheng. Real-time specification patterns. In *Proceedings of the 27th International Conference on Software Engineering*, ICSE '05, pages 372–381, New York, NY, USA, 2005. ACM. 34, 35

[72] G. Koutsandria, V. Muthukumar, M. Parvania, S. Peisert, C. McParland, and A. Scaglione. A hybrid network ids for protective digital relays in the power transmission grid. In *Smart Grid Communications (SmartGridComm), 2014 IEEE International Conference on*, pages 908–913, Nov 2014. 25

[73] S. Kriaa, L. Pietre-Cambacedes, M. Bouissou, et al. A survey of approaches combining safety and security for industrial control systems. *Reliability Engineering & System Safety*, 139:156 – 178, 2015. 112

[74] Marina Krotofil and Dieter Gollmann. Industrial control systems security: What is happening? *2013 11th IEEE International Conference on Industrial Informatics (INDIN)*, pages 664–669, 2013. ix, 3

[75] Christopher Kruegel, William Robertson, and Giovanni Vigna. Using alert verification to identify successful intrusion attempts. In *Practice in Information Processing and Communication (PIK 2004), 27(4):219 – 227, October*, 2004. 38

[76] J. Larsen. Breakage - Black Hat. https://www.blackhat.com/presentations/bh-dc-08/Larsen/Presentation/bh-dc-08-larsen.pdf, 2008. Access. : 2017-08. 50

[77] Laurens Lemaire, Jorn Lapon, Bart De Decker, and Vincent Naessens. A SysML Extension for Security Analysis of Industrial Control Systems. *2nd International Symposium for ICS & SCADA Cyber Security Research 2014 (ICS-CSR 2014)*, pages 1–9, 2014. 38

[78] Laurens Lemaire, Jan Vossaert, Joachim Jansen, and Vincent Naessens. Extracting vulnerabilities in industrial control systems using a knowledge-based system. In *Proceedings of the 3rd International Symposium for ICS & SCADA Cyber Security Research*, ICS-CSR '15, pages 1–10, Swinton, UK, UK, 2015. British Computer Society. 38

[79] C. Lemieux, D. Park, and I. Beschastnikh. General LTL specification mining. In *Proc. ASE'15*, pages 81–92, 2015. 36, 52, 56, 74

[80] Martin Leucker. Teaching runtime verification. In Sarfraz Khurshid and Koushik Sen, editors, *Runtime Verification*, pages 34–48, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. 33

[81] Martin Leucker. *Runtime Verification for Linear-Time Temporal Logic*, pages 151–194. Springer International Publishing, Cham, 2017. 32, 33

[82] Martin Leucker and Christian Schallhart. A brief account of runtime verification. *Journal of Logic and Algebraic Programming*, 78(5):293–303, 2009. 28

[83] Weize Li, Lun Xie, Zulan Deng, and Zhiliang Wang. False sequential logic attack on scada system and its physical impact analysis. *Computers & Security*, 58:149 – 159, 2016. 50

[84] Wenchao Li Wenchao Li, Alessandro Forin, and S.a. Sanjit a. Seshia. Scalable specification mining for verification and diagnosis. In *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, pages 755–760, 2010. 36, 52

[85] Hui Lin, Adam Slagell, Catello Di Martino, et al. Adapting Bro into SCADA: building a specification-based intrusion detection system for the DNP3 protocol. In *Proc. CSIIRW '13*, pages 1–4, 2013. 18

[86] Ondrej Linda, Todd Vollmer, and Milos Manic. Neural Network based Intrusion Detection System for critical infrastructures. *2009 International Joint Conference on Neural Networks*, pages 1827–1834, 2009. 21

[87] Z. Liu, C. Wang, and S. Chen. Correlating multi-step attack and constructing attack scenarios based on attack pattern modeling. In *2008 International Conference on Information Security and Assurance (isa 2008)*, pages 214–219, April 2008. 37

[88] Benjamin Livshits and Thomas Zimmermann. Dynamine: Finding common error patterns by mining software revision histories. In *Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ESEC/FSE-13, pages 296–305, New York, NY, USA, 2005. ACM. 36

[89] C. Lutz, F. Wolter, and M. Zakharyaschev. Temporal description logics: A survey. In *2008 15th International Symposium on Temporal Representation and Reasoning*, pages 3–14, June 2008. 30

[90] Oded Maler. Some thoughts on runtime verification. In *Runtime Verification - 16th International Conference, RV 2016, Madrid, Spain, September 23-30, 2016, Proceedings*, pages 3–14, 2016. 27, 49

[91] Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In Yassine Lakhnech and Sergio Yovine, editors, *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pages 152–166, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. 111

[92] Petr Matousek. Description and analysis of iec 104 protocol. Technical report, Brno University of Technology, Brno, Czech Republic, 2017. x, XII

[93] Metasploit. *http://www.metasploit.com/*. [Online, acc. March-2018]. 100

[94] Robert Mitchell and Ing-Ray Chen. Behavior Rule Specification-based Intrusion Detection for Safety Critical Medical Cyber Physical Systems. *IEEE Tran. on Depend. and Sec. Comp.*, 12(1):16–30, 2014. 25, 41, 42

[95] Robert Mitchell and Ing-Ray Chen. A survey of intrusion detection techniques for cyber-physical systems. *ACM Comput. Surv.*, 46(4):55:1–55:29, March 2014. 16

[96] Robert Mitchell and Ing-Ray Chen. Behavior rule specification-based intrusion detection for safety critical medical cyber physical systems. *Dependable and Secure Computing, IEEE Transactions on*, 12(1):16–30, Jan 2015. 22, 23, 24

[97] MITRE. Malicious control system cyber security attack case study - maroochy water services, australia. Technical report, MITRE, 2008. 10

[98] Yves Moinard. About the computation of forgetting symbols and literals. In *NMR 2006 (11th Workshop on Non-monotonic Reasoning)*, pages 209–217, Lake District, United Kingdom, 2006. Institut für Informatik, Technische Universität Clausthal, Germany (http://cig.in.tu-clausthal.de/). 89

[99] B. Morin and H. Debar. *Correlation of Intrusion Symptoms: An Application of Chronicles*, pages 94–112. Springer, 2003. 38

[100] Benjamin Morin, Ludovic Mé, Hervé Debar, and Mireille Ducassé. A logic-based model to support alert correlation in intrusion detection. *Information Fusion*, 10(4):285 – 299, 2009. Special Issue on Information Fusion in Computer Security. 37, 38

[101] Yosra Ben Mustapha, Hervé Débar, and Grégoire Jacob. Limitation of honeypot/honeynet databases to enhance alert correlation. In Igor Kotenko and Victor Skormin, editors, *Computer Network Security*, pages 203–217, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. 37, 38

[102] D. Nardella. Snap7. http://snap7.sourceforge.net, 2014. [Online, acc. : March-2018]. 97, 99

[103] HMS Industrial Networks. Network trends + technologies vision. http://ethercat.org/forms/italy2017/files/14_HMS_Trends_Technologies_Vision.pdf, 2017. [Online, acc. : March-2018]. 95, 97

[104] Peng Ning, Yun Cui, and Douglas S. Reeves. Constructing attack scenarios through correlation of intrusion alerts. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS '02, pages 245–254, New York, NY, USA, 2002. ACM. 39, 103

[105] Peng Ning, Yun Cui, Douglas S. Reeves, and Dingbang Xu. Techniques and tools for analyzing intrusion alerts. *ACM Trans. Inf. Syst. Secur.*, 7(2):274–318, May 2004. 36

[106] ODVA. The common industrial protocol (cip) and the family of cip networks. Technical paper, ODVA, Ann Arbor, Michigan, USA, 2016. x, XIII

[107] ODVA. *The Common Industrial Protocol (CIP) Specifications*. ODVA, 2016. XII

[108] ODVA. *The Ethernet/IP Specifications*. ODVA, 2016. XII

[109] Python OPC-UA. *https://github.com/FreeOpcUa/python-opcua*. [Online, acc. March-2018]. 100

[110] Alessandro Orso and Gregg Rothermel. Software testing: A research travelogue (2000–2014). In *Proceedings of the on Future of Software Engineering*, FOSE 2014, pages 117–132, New York, NY, USA, 2014. ACM. 28

[111] Shengyi Pan, Thomas Morris, and Uttam Adhikari. A Specification-based Intrusion Detection Framework for Cyber-physical Environment in Electric Power System. *International Journal of Network Security*, 17(2):174–188, 2015. 22

[112] Shengyi Pan, Thomas Morris, Senior Member, Uttam Adhikari, and Student Member. Developing a Hybrid Intrusion Detection System Using Data Mining for Power Systems. *IEEE Transactions on Smart Grid*, 6(6):3104–3113, 2015. 22

[113] Masood Parvania, Georgia Koutsandria, Vishak Muthukumary, Sean Peisert, Chuck McParland, and Anna Scaglione. Hybrid control network intrusion detection systems for automated power distribution systems. In *Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on*, pages 774–779, June 2014. 25, 26

[114] F. Pfenning. Automated theorem proving. https://cs.cmu.edu/~fp/courses/atp/handouts/atp.pdf, 2004. Lecture notes. 28

[115] Ludovic Pietre-Cambacedes, Yannick Fourastier, Fabrice Téa, Laurent Platel, David Boucart, Nicolas de Peslouan, Orion Ragozin, Patrice Bock, Jean-Claude Jabot, Pascal Sitbon, Marc Bouissou, Gérôme Billois, Pierre Kobes, Frédéric Guyomard, Stéphane Meynet, Thomas Demongeot, Frédéric Duflot, Mathieu Feuillet, and Thierry Lusseyran. *Cybersécurité des installations industrielles*. Cépaduès, 2015. 2, 5, 94

[116] David A. Plaisted. Theorem proving with abstraction. *Artificial Intelligence*, 16(1):47 – 108, 1981. 40

[117] Amir Pnueli. The temporal logic of programs. In *Proc. SFCS'77*, pages 46–57, Washington, DC, USA, 1977. IEEE Computer Society. 30

[118] Stanislav Ponomarev and Travis Atkison. Industrial Control System Network Intrusion Detection by Telemetry Analysis. *IEEE Transactions on Dependable and Secure Computing*, 5971(c):1–1, 2015. 21

[119] Dimitrie O Puaun and Marsha Chechik. On Closure Under Stuttering. *FAC*, 14:342–368, 2003. 31, 34

[120] PySMT. *http://github.com/pysmt/pysmt/*. [Online, acc. March-2018]. 100

[121] Jason Reeves, Ashwin Ramaswamy, Michael Locasto, Sergey Bratus, and Sean Smith. Intrusion detection for resource-constrained embedded control systems in the power grid. *International Journal of Critical Infrastructure Protection*, 5(2):74–83, 2012. 21

[122] Chaiman Robert T. Marsh. Critical foundations: Protecting america's infrastructures. Technical report, The Report of the President's Commission on Critical Infrastructure Protection, October 1997. 50, 67

[123] Sebastian Roschke, Feng Cheng, and Christoph Meinel. A new alert correlation algorithm based on attack graph. In Álvaro Herrero and Emilio Corchado, editors, *Computational Intelligence in Security for Information Systems*, pages 58–67, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. 38

[124] Julian Rrushi and Kyoung-Don Kang. Detecting Anomalies in Process Control Networks. *IFIP Advances in Information and Communication Technology*, 311:151–165, 2009. 21, 22

[125] R. Sadoddin and A. Ghorbani. Alert correlation survey: Framework and techniques. In *Proc. of the 2006 Int. Conf. on Privacy, Security and Trust*, PST '06, pages 37:1–37:10, NY, USA, 2006. ACM. 37

[126] L. Saitta and J-D Zucker. *Abstraction in artificial intelligence and complex systems*. Computer Science. Springer, 2013. 40, 41, 78

[127] Saeed Salah, Gabriel Maciá-Fernández, and Jesús E. Díaz-Verdejo. A model-based survey of alert correlation techniques. *Computer Networks*, 57(5):1289 – 1317, 2013. 37

[128] J. Schumann, P. Moosbrugger, and K.Y. Rozier. R2U2 : Monitoring and Diagnosis of Security Threats for Unmanned Aerial Systems. In *6th Int. Conf. Runtime Verification (RV'15), Vienna, Austria*, pages 233–249, 2015. 24, 25, 42

[129] U. Shankar and V. Paxson. Active mapping: resisting nids evasion without altering traffic. In *2003 Symposium on Security and Privacy, 2003.*, pages 44–61, May 2003. 37, 38

[130] R. Sommer and V. Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *2010 IEEE S&P*, pages 305–316, 2010. 41

[131] Fu Song and Zhilin Wu. On temporal logics with data variable quantifications: Decidability and complexity. *Information and Computation*, 251:104 – 139, 2016. 29, 30

[132] Siddharth Sridhar and Manimaran Govindarasu. Model-based attack detection and mitigation for automatic generation control. *IEEE Transactions on Smart Grid*, 5(2):580–591, 2014. 24, 25, 49

[133] K. Stouffer, J. Falco, and K. Scarfone. *SP 800-82 Rev 2. Guide to Industrial Control Systems (ICS) Security.* NIST, 2015. 2, 3, 5, 6

[134] Kleber Stroeh, Edmundo Roberto Mauro Madeira, and Siome Klein Goldenstein. An approach to the correlation of security events based on machine learning techniques. *Journal of Internet Services and Applications*, 4(1):7, Mar 2013. 37

[135] A. Swales. Open modbus/tcp specification. Standard, Schneider Electric, 1999. IX

[136] TC 57 Power systems management and associated information exchange. *IEC 60870-5-104:2006.* International Electrotechnical Commission (IEC), 2006. XI

[137] R. Turton. *Analysis, Synthesis, and Design of Chemical Processes.* Prentice-Hall international series in engineering. Prentice Hall, 2012. 65

[138] US-CERT. Crashoverride. https://www.us-cert.gov/ncas/alerts/TA17-163A, 2017. [Online, acc. : March-2018]. xiii, 12, 25, 109

[139] A. Valdes and K. Skinner. Probabilistic alert correlation. In *Int. Workshop on Recent Advances in Intrusion Detection*, pages 54–68. Springer, 2001. 26, 37, 38, 77, 97

[140] F. Valeur, G. Vigna, C. Kruegel, et al. Comprehensive approach to intrusion detection alert correlation. *IEEE Transactions on Dependable and Secure Computing*, 1(3):146–169, July 2004. 26, 37, 39, 77, 78, 81, 102

[141] Moshe Y. Vardi. An automata-theoretic approach to linear temporal logic. *Banff Higher Order Workshop 1995*, 1996. 33

[142] Yong Wang, Zhaoyan Xu, Jialong Zhang, Lei Xu, Haopei Wang, and Guofei Gu. SRID : State Relation Based Intrusion Detection for False Data Injection Attacks in SCADA. *Computer Security - ESORICS 2014*, 8173:401–418, 2014. 22, 24

[143] A. Wedgbury and K. Jones. Automated asset discovery in industrial control systems: Exploring the problem. In *Proc. of the 3rd Int. Symp. for ICS & Scada Cyber Security Research*, pages 73–83, Swindon, UK, 2015. BCS. 95, 99

[144] Westley Weimer and George C. Necula. Mining temporal specifications for error detection. In Nicolas Halbwachs and Lenore D. Zuck, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 461–476, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. 36

[145] Joseph Weiss and Joseph Weiss. *Protecting Industrial Control Systems from Electronic Threats.* Momentum Press, 2010. 2

[146] Dayu Yang, Alexander Usynin, and Jw Hines. Anomaly-based intrusion detection for SCADA systems. In *5th Intl. Topical Meeting on Nuclear Plant Instrumentation, Control and Human Machine Interface Technologies (NPIC&HMIT 05)*, pages 12–16, 2005. 21

[147] Jinlin Yang, David Evans, Deepali Bhardwaj, et al. Perracotta: Mining temporal api rules from imperfect traces. In *Proc. ICSE '06*, pages 282–291, New York, NY, USA, 2006. ACM. 36, 52

[148] M. Yoon and G. Ciocarlie. Communication Pattern Monitoring : Improving the Utility of Anomaly Detection for Industrial Control Systems. In *SENT*, 2014. 18, 19, 22, 25, 41

[149] Ö. Yüksel, J. den Hartog, and S. Etalle. Reading between the fields: Practical, effective intrusion detection for industrial control systems. In *Proc. of the 31st Annual ACM Symp. on Applied Computing*, SAC '16, pages 2063–2070, NY, USA, 2016. ACM. 18, 25, 42, 100

[150] Zeep. *https://github.com/mvantellingen/python-zeep*. [Online, acc. March-2018]. 100

[151] Lichen Zhang. Multi-view approach to specify and model aerospace cyber-physical systems. In *Computational Science and Engineering (CSE), 2013 IEEE 16th International Conference on*, pages 595–602, Dec 2013. 16

[152] C. Zhou, S. Huang, N. Xiong, et al. Design and analysis of multimodel-based anomaly intrusion detection systems in industrial process automation. *IEEE Trans. Systems, Man, and Cybernetics: Systems*, 45(10):1345–1360, 2015. 16, 25, 26

[153] Bonnie Zhu and Shankar Sastry. SCADA-specific Intrusion Detection/Prevention Systems: A Survey and Taxonomy. In *Proceedings of the First Workshop on Secure Control Systems (SCS '10)*, 2010. 16

[154] Christopher Zimmer, Balasubramany Bhat, et al. Time-based intrusion detection in cyber-physical systems. In *Proc. First ACM/IEEE Int. Conf. on CPS*, pages 109–118, 2010. 21, 22

[155] Tanja Zseby, Benoit Claise, Juergen Quittek, and Sebastian Zander. Requirements for IP Flow Information Export (IPFIX). RFC 3917, October 2004. 16