SPECIFICATION-BASED PREDICTIVE CONTINUOUS MONITORING FOR CYBER PHYSICAL SYSTEMS WITH UNOBSERVABLES

alessio coletta

Ph.D.
Department of Information Engineering and Computer Science
University of Trento

October 2018

Supervisor: Alessandro Armando

Sapiens fingit fortunam sibi.

A wise man carves his own fortune.

— Plautus

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACRONYMS

CN	Corporate Network
CPS	Cyber Physical System
DCS	Distributed Control System
DMZ	Demilitarised Zone
DNF	Disjunctive Normal Form
HMI	Human Machine Interface
ICS	Industrial Control System
ICT	Information and Communication Technology
IED	Intelligent Electronic Device
IOT	Internet of Things
LAN	Local Area Network
LP	Linear Programming
NIST	National Institute of Standards and Technology
PLC	Programmable Logic Controller
PPL	Parma Polyhedra Library
RMF	NIST Risk Management Framework

RTU	Remote Terminal Unit
SCADA	Supervisory Control And Data Acquisition
SMT	Satisfiability Modulo Theories
VPN	Virtual Private Network

Part I

BACKGROUND

1 INTRODUCTION

1.1 MOTIVATIONS

Cyber Physical System (CPS) are composed by networked ICT devices that support the operation of physical entities. In this work CPS is a general term that includes Industrial Control Systems (ICS), building automation systems, and the Internet of Things (IoT) used for control and automation. Such systems are employed in a large number of critical infrastructures, even life-critical ones. Despite their criticality, CPS often lack appropriate cyber security measures.

Until a couple of decades ago, industrial control systems used to employ serial-based proprietary communication protocols designed with no security requirements. However, plants and installations were completely isolated. The progressive use of ICT technology exposed ICS to vulnerabilities and threats typical of the ICT world. Moreover, the plants and installations became more and more interconnected for remote maintenance / control / planning, in order to ease operations and to decrease management costs. Unfortunately, the cyber security counterpart did not improved accordingly.

CPS present many specific differences from standard ICT systems that make general ICT security solutions seldom effective in this context [102]. For instance, change management is particularly difficult or even impossible, and the lifetime of devices is much longer than typical ICT systems, often measured in decades. As such, existing CPS are usually characterised by legacy components.

The first attempts to increase the security level of ICS were targeted at developing secure versions of control network protocols and devices. However, although effective solutions have been designed and implemented, replacing legacy components in ICS is hardly a viable option. For this reason, in recent years cyber security experts, both in the academia and in the industry, have progressively shifted their efforts from the robustness approach to the resilience one. In other words, instead of trying to replace legacy components of existing installations with more secure ones, the goal is to improve the capability to detect possible security issues and to react as promptly and accurately as possible. The increasing availability of security monitoring tools and the establishment of international computer response practices (e.g. CERT / CSIRT) supported this approach. As a consequence, situational awareness and monitoring have recently become a key for improving the cyber security resilience of ICS.

Besides ICS, recent years show an increasing trend in the IoT sector. Billions of devices are expected to be connected in the next future and cyber security experts have began to point out possible threats and the expected wide attack surface. It is difficult to make predictions about the actual risks related to the IoT, but specific cyber security monitoring approaches are likely to be an essential tool for situational awareness and effective incident response in this sector.

Although CPS specific differences make protection solutions less effective, some of these peculiarities can also lead to better tailored monitoring solutions. Indeed, CPS are typically designed for a specific purpose in a predetermined environment. As a consequence, the behaviour of their physical process is predictable to a good extent and often well documented. Hence, it is possible to leverage such knowledge of the CPS to specify known critical conditions. It is also possible to combine cyber and process aspects for a greater expressiveness and effectiveness. However, to our best knowledge, specification-based security monitoring approaches appear less mature than other approaches like anomaly-based techniques. This work presents a contribution in this regard.

The main assumption of the whole work is that an operator is able to grow some knowledge about the system to be monitored and to identify some conditions that he/she considers critical. In this setting, the term *critical* means any illicit, unwanted, anomalous, or suspicious behaviour that the operator is willing to detect. This knowledge may arise from documentation, from experience, from surveys and interviews, from clustering techniques, from machine learning-based analysis approaches, or any other source. This work makes no assumption on the kind and the source of this information. The assumption is that experts of the CPS are able to:

1. identify the aspects of interest of the CPS to observe;
2. express critical conditions on top of these aspects.

It is necessary to observe the current state of the CPS to detect if it has reached a critical state. In most occasions, when the CPS reaches a critical state, it is already too late. It is beneficial to have a way to predict, to some extent, if the system is evolving toward critical conditions. This work contributes in this regards.

Moreover, in real cases the assumption that the aspects of interest are always observable is too strict. A monitoring framework should be able to deal with unobservable variables to be applicable to a wider range of real cases. This work addresses both predictiveness and unobservables.

This work aims at developing a novel and effective approach for continuous monitoring suitable for Security Operation Centres (SOC) and Computer Security Incident Response Teams (CSIRT) of enterprises that largely employ CPS.

The first contribution is a thorough literature review. The review started from the technical aspects of the cyber components of CPS, the legacy and present technologies, their security issues, and possible solutions. This includes both academic literature, which explores the state of the art of CPS cyber security, and international standards and guidelines, aimed at fostering appropriate risk management methodologies for critical infrastructures.

My personal ten years working experience in CPS cyber security has strengthened my understanding of the sector. This allowed me to carry out my industrial PhD activities within a privileged environment. The active participation in European projects, in the context of FP7/Horizon 2020, targeting the security of the energy sector and Smart Grids, allowed me to have continuous discussions with European stakeholders like Enel (Italy), Alliander (The Netherlands), Efacec (Portugal), EDP (Portugal), EVN (Austria), ENISA, the EU Joint Research Centre, RSE (Italy), ENCS (The Netherlands), SecurityMatters (The Netherlands), Deloitte (Italy), Ansaldo Energia (Italy), and others. My current Manufacturing and Automotive Cyber Security position at Magneti Marelli gives me the opportunity to see how human plant operators are an inherent component in the loop and a valuable asset to leverage essential knowledge.

The literature review and the working experience prove that solutions for improving the resilience of CPS are necessary. In this respect, incident response teams are considered a fundamental component of risk management and continuous monitoring and attack detection are necessary to support such methodologies. The literature shows that leveraging the peculiarities of CPS leads to more effective techniques. Most of the works focus on a particular combination of observations and analysis of features, both from the cyber and the physical worlds, often presenting unsupervised or semi-supervised techniques based on statistics or machine learning. Specification-based approaches appear less mature.

The aim of this work is to develop a security monitoring framework for CPS which is based on specifications by human domain experts. The framework allows to define the features of the CPS to be observed and the critical condition to be detected. Unlike intrusion detection systems, which are limited to sending an alert when a suspicious activity is detected, our framework presents a quantitative notion of criticality of the monitored system with respect to a critical condition. It allows to continuously evaluate such criticality and to track how it changes in time, providing a way to predict whether the system is evolving to a critical or licit states.

The framework is able to handle unobservable features, i.e. variables that are necessary to express a critical condition but whose value is not available, temporarily or permanently. Broken sensors or human intentions are examples of unobservable variables. This capability appears completely novel in the literature.

1.2 PRELIMINARIES AND CONVENTIONS

LINEAR ALGEBRA AND CONVEX POLYHEDRA. This section presents notations and concepts derived from [1]. We denote by $\mathbb{R}^n$ the n-dimensional vector space on the field of real numbers $\mathbb{R}$, endowed with the standard topology. The set of all non-negative reals is denoted by $\mathbb{R}^+$. For each $i \in \{1, \dots, n\}$, v_i denotes the i-th component of the (column) vector $v = (v_1, \dots, v_n)^T \in \mathbb{R}^n$. We denote by 0 the vector of R^n, called the origin, having all components equal to zero. A vector $v \in \mathbb{R}^n$ can be also interpreted as a matrix in $\mathbb{R}^{n \times 1}$ and manipulated accordingly using the usual definitions for addition, multiplication (by a scalar and by another matrix), and transposition denoted by v^T.

The dot product (or scalar product or inner product) of $v, w \in \mathbb{R}^n$ is $v \cdot w = v^T w = \sum_i v_i w_i$.

Definition 1.1. For any $S_1, S_2 \subseteq \mathbb{R}^n$, the Minkowski's sum of S_1 and S_2 is defined as $S_1 + S_2 = \{v_1 + v_2 \mid v_1 \in S_1, v_2 \in S_2\}$.

Definition 1.2. For each vector $a \in \mathbb{R}^n$ and scalar $b \in \mathbb{R}$, where $a \neq 0$, and for each relation symbol $\bowtie \in \{=, \leq, <, \geq, >\}$, the linear constraint $a^T x \bowtie b$ on vectors $x \in R^n$ defines:

- an affine hyperplane if it is an equality constraint, i.e $\bowtie \in \{=\}$;
- a closed affine half-space if it is a non-strict inequality constraint, i.e. $\bowtie \in \{\leq, \geq\}$;
- an open affine half-space if it is a strict inequality constraint, i.e. $\bowtie \in \{<, >\}$.

Definition 1.3. The set $P \subseteq \mathbb{R}^n$ is a *convex polyhedron* if and only if either P can be expressed as the intersection of a finite number of (open or closed) affine half-spaces of $\mathbb{R}^n$ or $n = 0$ and $P = \emptyset$. The set of all convex polyhedra on the vector space $\mathbb{R}^n$ is denoted $\mathbb{P}_n$.

Definition 1.4. The set $P \in \mathbb{P}_n$ is a *closed* convex polyhedron if and only if either P can be expressed as the intersection of a finite number of closed affine half-spaces of $\mathbb{R}^n$ or $n = 0$ and $P = \emptyset$. In this work we use only closed polyhedron.

Convex polyhedra can be specified by using two representations: the linear constraints representation and the generators one.

Definition 1.5 (Constraints representation)**.** By definition, each polyhedron $P \in \mathbb{P}_n$ is the set of solutions to a constraint system, i.e. a finite number of constraints. By using matrix notation, we have

$$P = \{x \in \mathbb{R}^n \mid A_1 x = b_1, A_2 x \leq b_2, A_3 x < b_3\} \tag{1.1}$$

where, for $i \in \{1, 2, 3\}$, $A_i \in \mathbb{R}^{m_i} \times \mathbb{R}^n$ and $b_i \in \mathbb{R}^{m_i}$, and $m_1, m_2, m_3 \in \mathbb{N}$ are the number of equalities, the number of non-strict inequalities, and the number of strict inequalities, respectively. As this work only employees closed polyhedra, the third element $A_3 x < b_3$ of strict inequalities is not used, i.e. $m_i = 0$.

Definition 1.6 (Combinations and Hulls)**.** Let $S = \{x_1, \ldots, x_k\} \subseteq \mathbb{R}^n$ be a finite set of vectors. For any set of k scalars $\lambda_1, \ldots, \lambda_k \in \mathbb{R}$, the vector $v = \sum_{j=1}^{k} \lambda_j x_j$ is said to be a linear combination of the vectors in S. Such a combination is said to be

- a positive (or conic) combination if $\forall j \in 1, \ldots, k.\, \lambda_j \in \mathbb{R}^+$;
- an affine combination if $\sum_{j=1}^{k} \lambda_j = 1$;
- a convex combination if it is both positive and affine.

We denote by $\text{linearhull}(S), \text{conichull}(S), \text{affinehull}(S), \text{convexhull}(S)$ the set of all the linear, positive, affine, convex combinations of the vectors in S.

Definition 1.7. Let $P \in \mathbb{P}_n$ be a convex polyhedron. Then

- a vector $p \in P$ is called a *point* of P;
- a vector $r \in \mathbb{R}^n$, where $r \neq 0$, is called a *ray* of P if $P \neq \emptyset$ and $p + \lambda r \in P$ for all points $p \in P$ and all $\lambda \in \mathbb{R}^+$;
- a vector $l \in \mathbb{R}^n$ is called a *line* of P if both l and $-l$ are rays of P.

A point of a polyhedron $P \in \mathbb{P}_n$ is a *vertex* if and only if it cannot be expressed as a convex combination of any other pair of distinct points in P. A ray r of a polyhedron P is an extreme ray if and only if it cannot be expressed as a positive combination of any other pair r_1 and r_2 of rays of P, where $r \neq \lambda r_1, r \neq \lambda r_2$ and $r_1 \neq \lambda r_2$ for all $\lambda \in \mathbb{R}^+$ (i.e., rays differing by a positive scalar factor are considered to be the same ray). Notice that this definition is stricter than linear independence.

Definition 1.8 (Generators representation)**.** Each closed polyhedron $P \in \mathbb{P}_n$ can be represented by finite sets of lines L, rays R, and points P. The 3-tuple $G = (L, R, P)$ is said to be a generator system for P, in the sense that

$$P = \text{linear.hull}(L) + \text{conic.hull}(R) + \text{convex.hull}(P) \tag{1.2}$$

where the symbol $+$ denotes the Minkowski's sum in Definition 1.1.

Any closed polyhedron P can be described by using a constraint system C, a generator system G, or both by means of the double description pair (C, G). Given one kind of representation, there are algorithms for computing a representation of the other kind and for minimising both representations by removing redundant constraints/generators.

The Parma Polyhedra Library (PPL) [1] provides double description. In this work we use the he following PPL functions[1]:

1 The real function names differ. We show a simplification for clarity purpose.

- ppl.from_constraints(C): given a set of (closed) linear constraints C, it returns the corresponding polyhedron.
- ppl.from_generators(P, L, R): given a set of points P, a set of lines L, and a set of rays R, it returns the corresponding polyhedron.
- P.get_constraints(): given a PPL polyhedron P, it returns the set of linear constraints C.
- P.get_generators(): given a PPL polyhedron P, it returns the set of minimised generators (V, L, R) where V is the set of vertices, R the set of rays, and L the set of lines.
- P.contains(Q): given two (closed) polyhedra P and Q, it returns true if $Q \subseteq P$.
- P.intersects(Q): given two (closed) polyhedra P and Q, it returns true if $P \cap Q \neq \emptyset$.

METRIC SPACES

Definition 1.9. Given a set X, a function $d\colon X \times X \to \mathbb{R}$ is called *premetric* if both $d(x, y) \geq 0$ and $d(x, x) = 0$ for all $x, y \in X$. Given a set X, a premetric function $d\colon X \times X \to \mathbb{R}$ is called a *metric* if for all $x, y, z \in X$:

1. $d(x, y) = 0$ iff $x = y$,
2. $d(x, y) = d(y, x)$,
3. $d(x, y) \leq d(x, z) + d(z, y)$.

The pair (X, d) is called *metric space*.

We use the following well known result.

Proposition 1.1. *Let (X, d) be a metric space. The function $D\colon 2^X \times 2^X \to \mathbb{R}$ defined as*

$$D(A, B) = \inf_{a \in A,\, b \in B} d(a, b) \tag{1.3}$$

is a premetric.

By abusing notation, $D(a, B)$ is used to denote $D(\{a\}, B)$ when clear from the context.

Definition 1.10 (Manhattan distance)**.** Let $d\colon \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^+$ defined as

$$d(x,y) = \sum_{i=1}^{n} |x_i - y_i| \tag{1.4}$$

Equivalently, $d(x,y) = |x - y|$ where $|\cdot|$ denotes the standard l_1-norm on $\mathbb{R}^n$. The function d is a metric on $\mathbb{R}^n$ and is also referred as Manhattan distance, taxicab metric, or L_1 distance.

2 CYBER PHYSICAL SYSTEMS

2.1 OVERVIEW

This chapter describes the main features of CPS. Section 2.2 presents Industrial Control System (ICS), their components, and the main cyber security issues. Section 2.3 presents a short introduction to the Internet of Things (IoT) and a brief overview of their cyber security problems.

2.2 INDUSTRIAL CONTROL SYSTEMS

This section introduces ICS, summarising their main components and functionalities. ICS are very heterogeneous systems, and each industrial installation is unique. While a complete description and characterisation of industrial system is out of the scope of the present work, it is important to define the main common components.

A general description of ICS together with a possible *ICS reference architecture* is presented. The reference architecture precisely defines which are the ICS components of interest to our aim. Despite being high level, this architecture presents all the major ICT components of most ICS.

After introducing ICS and the reference architecture, an analysis of information security issues of ICS is presented. Our monitoring framework is focused on the cyber security of the networked components.

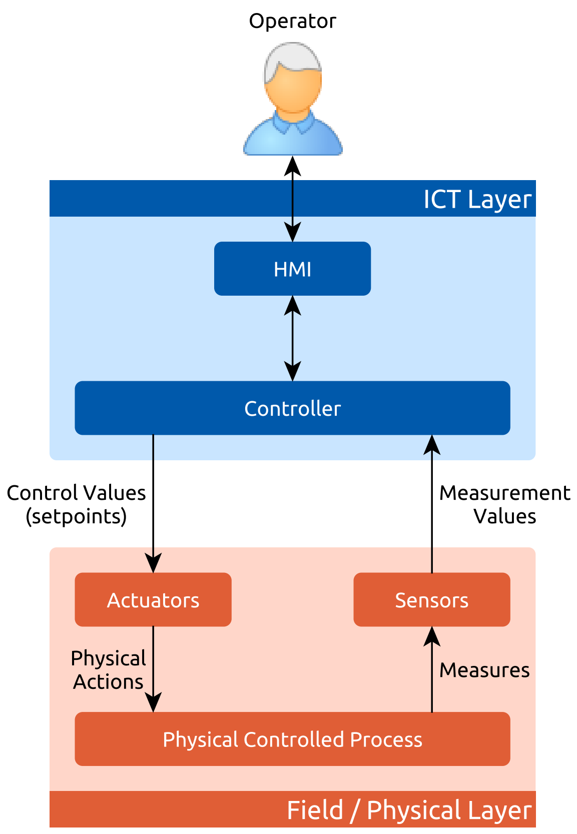


Figure 2.1: Simple Schema of Control Loop.

According to the US National Institute of Standards and Technology (NIST) [102], ICS is a general term to indicate several kinds of systems for controlling industrial processes, including Supervisory Control And Data Acquisition (SCADA) systems and Distributed Control System (DCS), or any system that includes Programmable Logic Controller (PLC) to control physical entities. ICS are used in different industrial sectors, ranging on water, gas, and oil transportation, chemical industries, goods manufacturing, logistics, power generation, healthcare, etc.

As a consequence it is not surprising that ICS are different from each other, and each industrial installation is unique. Nevertheless, typical ICS shares the same core concepts and features. The main concept behind a control system is the *control loop*, depicted in Figure 2.1.

The aim of an ICS is to control a physical process. In the physical layer a set of *field devices* enables the actual control of the physical process. Field devices are either *actuators* or *sensors*.

Actuators are devices that have a direct impact on the process (e.g. pumps, motors, cooling fans). An actuator is usually characterised by a set of parameters that alter the way it interacts with the physical process. For instance, a cooling fan is characterised by its rotational speed (rpm). The set of values characterising the actuators is often called setpoints. The primary way to control the physical process is altering the relevant setpoints (e.g. increasing or decreasing the rpm of the cooling fan). The actuator will alter the process in response of changes to setpoints.

On the other hand, a sensor is a device that measures parameters of interest of the physical process (e.g. a thermometer). Sensors are fundamental to correctly supervise and operate the process. In general, the correct actual setpoints to send to actuators are the results of control logic whose inputs are measurements from sensors.

In other words, field devices enable full control of the process by reading values from sensors and altering the setpoints of the actuators. An industrial process can be very complex and composed by hundreds or thousands of field devices. For this reason, it is crucial to automate the data gathering from field device and to implement a the control logic to adjust the setpoints accordingly both to physical changes and to the intervention of human operators. The *controller* box in Figure 2.1 represents an ICT system aimed at this. It is worth noticing that the controller box in the Figure may be implemented by a complex system that employs different components often geographically distributed.

The Human Machine Interface (HMI) box in Figure 2.1 represents the ICT component that shows the human operator the current set points and the sensor values of the controlled process. Moreover, the HMI enables an operator to change the setpoint.

While the control loop is the abstract main concept behind any ICS, the real complexity is hidden in the controller, which is a complex system of ICT components. A portion of these components interacts with the field device to "tie the loop". The NIST Special Publication 800-82 [102] enumerates typical components of the ICT layer of an industrial control system. The following definitions are quoted from that document.

CONTROL SERVER. The control server hosts the DCS or PLC supervisory control software that communicates with lower-level control devices. The control server accesses subordinate control modules over an ICS network.

SCADA SERVER OR MASTER TERMINAL UNIT. The SCADA Server is the device that acts as the master in a SCADA system. Remote Terminal Units and PLC devices (as described below) located at remote field sites usually act as slaves.

REMOTE TERMINAL UNIT. The Remote Terminal Unit (RTU), also called a remote telemetry unit, is a special purpose data acquisition and control unit designed to support SCADA remote stations. RTUs are field devices often equipped with wireless radio interfaces to support remote situations where wire-based communications are unavailable. Sometimes PLCs are implemented as field devices to serve as RTUs; in this case, the PLC is often referred to as an RTU.

PROGRAMMABLE LOGIC CONTROLLER. The PLC is a small industrial computer originally designed to perform the logic functions executed by electrical hardware (relays, switches, and mechanical timer/counters). PLCs have evolved into controllers with the capability of controlling complex processes, and they are used substantially in most SCADA systems and DCS. Other controllers used at the field level are process controllers and RTUs; they provide the same control as PLCs but are designed for specific control applications. In SCADA environments, PLCs are often used as field devices because they ar more economical, versatile, flexible, and configurable than special-purpose RTUs.

INTELLIGENT ELECTRONIC DEVICES (IED). An IED is a "smart" sensor or actuator containing the intelligence required to acquire data, communicate to other devices, and perform local processing and control. An Intelligent Electronic Device (IED) could combine an analog input sensor, analog output, low-level control capabilities, a communication system, and program memory in one device. The use of IEDs in SCADA and DCS systems allows for automatic control at the local level.

HUMAN-MACHINE INTERFACE. The HMI is software and hardware that allows human operators to monitor the state of a process under control, to modify control settings to change the control objective, and to manually override automatic control operations in the event of an emergency. The HMI also displays process status information, historical information, reports, and other information to operators, administrators, managers, business partners, and other authorized users. The location, platform, and interface may vary. For example, an HMI could be a dedicated platform in the control centre, a laptop on a wireless LAN, or a browser on any system connected to the Internet.

DATA HISTORIAN. The data historian is a centralized database for logging all process information within an ICS. Information stored in this database can be accessed to support various analyses, like troubleshooting statistical process control, and enterprise level planning.

INPUT/OUTPUT (IO) SERVER. The IO server is a control component responsible for collecting, buffering and providing access to process information from control sub-components such as PLCs, RTUs and IEDs. An IO server can reside on the control server or on a separate computer platform. IO servers are also used for interfacing third-party control components, such as an HMI and a control server.

Previous definitions cover most of the families of the main ICT components of ICS. However, they reflect an heterogeneous world characterised sometimes by redundant and overlapping components and terminology. In order to capture the common aspect of interest of ICS with respect to cyber security, it is important to define a reference set of components and a reference architecture that models the vast majority of industrial control systems.

Figure 2.2 depicts the main components typical of an industrial control system, and shows their architecture and connections. This work focuses on cyber threats of networked CPS, spanning on the cyber and the process aspects and their relation.

The ICT system of industrial companies is usually very big, and serves many duties and purposes. The whole IT infrastructure can be divided in several sub-networks. One or more firewalls (represented by a single firewall in Figure 2.2) keeps the networks segregated by a proper set of rules. Network segregation rules are usually part of the security procedures resulting from the *Information Security Management* and from the IT departments of the company, and reflect norms and best practices in the field.

The Corporate Network (CN) serves all the common IT services required by any organisation. This network encompasses all the office workstations, printers, laptops, and some corporate servers. Usually workstations in the corporate network are not supposed to directly interact with the control system. However, it is likely to find operat-

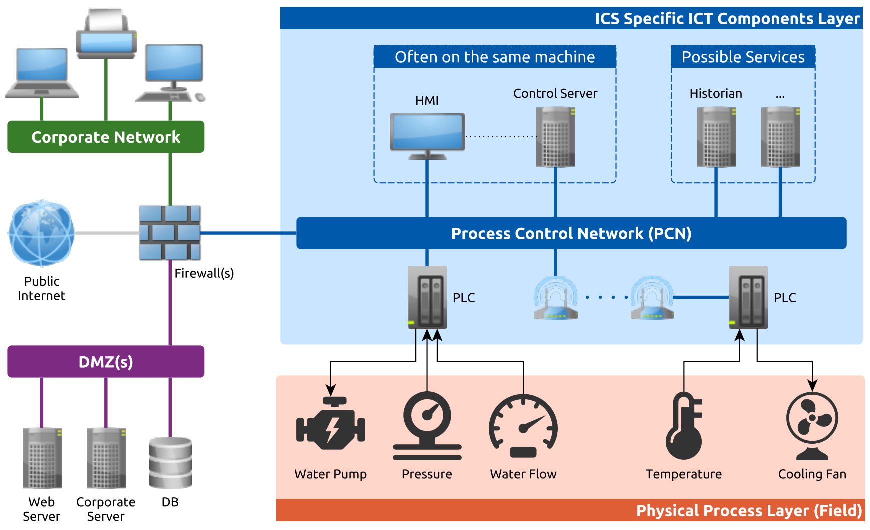


Figure 2.2: ICS Architecture Reference Schema.

ors' using their office workstation as a terminal to operate the process. In the real world this is usually achieved through Virtual Private Networks (VPN) to some HMI. This is a common practice even if this is often discouraged by several international security guidelines.

Critical services are usually hosted on servers inside Demilitarised Zones (DMZ). A DMZ is a common configuration for enforcing perimetral security on logical sub-networks containing critical services like web servers, mail servers, proxies.

The core ICT part of an industrial control network sits in the *Process Control Network*. PCN contains all the ICT components that implement the ICT layer of the loop. The *Control Server* (CS) is the most important component. It is able to send commands to the PLCs for gathering information about the process (i.e. reading sensor values) and to alter the process (i.e. to change the set points of the actuators).

The HMI can be a specific hardware device connected to the CS, showing set points and sensor values, and allowing to alter set points. Sometimes an HMI is just a function of the Control Application running on a control server (if it has input and output devise like a keyboard and a display).

PLCs are hardware devices interfacing the ICT layer with the physical process layer. From one side, a PLC is connected to the PCN, usually via Ethernet cables of with any kind of wireless communication. The PCN side enables the PLC to exchange network messages with the control server. On the other side, a PLC is connected to one or more field devices (sensor or actuator) through electric wires. This enables the PLC to actually read sensors' measurements and to operate the actuators. PLC can be programmed using simple logics (called ladder logic) in order to automate certain operations. However, this project is targeted to information security aspects, and PLC logics are not considered.

The network boxes (Corporate, DMZ, and PCN) represent any subnetwork configuration, using switches, routers, virtual LANs, and so on. Network connections can be wired or wireless.

2.2.1 *Cyber Security Aspects*

As mentioned before, originally ICS used to be isolated systems based on ad-hoc hardware products communicating through proprietary protocols on cable connections, typically serial cables or similar. Thus, ICS where inherently secure from external cyber attacks. For the same reason, information security was not considered as a core aspect of the design, development, and deployment of ICS.

Information security solutions typical of IT systems apply to the industrial sector only to some extent. Indeed, even if many principle are perfectly suitable for ICS, their application may be totally different. This Section presents an analysis of information security issues of industrial control systems, and some differences with regular IT systems.

High level security needs in term of *Confidentiality, Integrity, and Availability* (CIA), which are often enumerated in this order of importance for IT systems, assume a different meaning in industrial systems. Indeed, Availability is probably the most important, closely followed by Integrity. Confidentiality, on the other hand, is usually less important. In some ICS documents and documentations it is easy to find the acronym AIC instead of CIA.

It is crucial to examine some peculiar aspects of ICS in order to understand to what extent the AIC point of view influence the whole information security picture. Safety, in term of human lives, play a big role in industrial systems. Availability is a crucial requirement for guaranteeing that controlled processes behaves correctly and that human operators are safe. Moreover, in the industrial sector interrupting a service, even for few hours, may have a huge financial impact on the organisation (e.g. in the case of electric power outages). Hence, availability is top priority for a 24/7 system for controlling a possibly dangerous physical process.

As a consequence of this different point of view from standard ICT systems, in terms of information security, most risk assessment methodologies must be calibrated for the industrial sector. Indeed, while the same Information System Management principles still apply [15], the actual implementation of those principle may be completely different. Risk assessment must take into account that ICT components of ICS must interact with a physical environment, and the methodology for assessing the assets' security levels may require a higher level of confidence of the risk assessed. For instance, while for typical ICT certain risks can be accepted or transferred, for industrial system often this is not the case, because threats are impacting directly on human lives or can disrupt the whole industrial process. Similarly, business continuity and disaster recovery are crucial in the industrial sector, even more than in typical IT system.

Performance aspects have a big role in information security for industrial systems. ICS are often based on simple logic algorithms implemented on thousands of devices which have a small computational power, are geographically distributed, and communicate on heterogeneous network (also mobile networks). Nevertheless, the need for high availability imposes strict real-time requirements. In such a setting, every security measure can easily have an overheard bigger than the effectiveness of the measure. In other words, securing industrial control system can be very difficult because any additional measure can alter the whole system to a point that real-time requirements are not satisfied.

Change management is a good example of this. In typical ICT settings patch management is a crucial aspect of any security management system. An out of date system is probably the most likely entry point of an attack, because it presents known and easy to expoit vulnerabilities. In a sentence, ICT components have to be always updated. This is not true for industrial control systems. First, specialised hardware manufactures provide strict security level agreements and are subject of precise terms of liability. For this reason they try to keep their hardware under control, not allowing customers (industrial organisations) to arbitrarily deploy any security software. An industrial company usually can only update their systems only with patches that are approved by vendors, slowing down the whole update process. Moreover, only redundant systems can be easily shut down, updated, and replaced, and sometimes software updates are difficult and require costly down-times. In other words, the industrial sector is inherently characterised by very high percentage of unpatched and obsolete software.

While availability is usually considered the most important aspect of ICS, integrity is crucial as well. Clearly, corrupted data do not make the system work correctly. From this point of view, data integrity of communication is one of the core aspects for any industrial control system, as each components keep communicating data that support the correctness of the whole process. However, most of the current widely used network protocols are a mere port of the old ones on top of TCP/IP, without concerns about information security. Moreover, the lifetime of ICS components is longer than common IT components, as some systems keep working for 15-20 years or more. Thus, many ICS currently use obsolete protocols without security mechanisms, instead of correct and secure protocols with a proper and modern design from scratch. As a consequence, many ICS are inherently insecure and many vulnerability exists in current installations.

The National Institute of Standards and Technology (NIST) presents a detailed documents about information security of ICS [102]. Among other relevant considerations, it contains an analysis of technical differences between regular IT systems and ICS. The following list is taken from that document.

IT systems	**ICS**
Performance Requirements	
Non-real-time	Real-time
Response must be consistent	Response is time-critical
High throughput is demanded	Modest throughput is acceptable
High delay and jitter may be acceptable	High delay and/or jitter is not acceptable
Availability Requirements	

Responses such as rebooting are acceptable Availability deficiencies can often be tolerated, depending on the system's operational requirements	Responses such as rebooting may not be acceptable because of process availability requirements Availability requirements may necessitate redundant systems Outages must be planned and scheduled days/weeks in advance High availability requires exhaustive pre- deployment testing
Risk Management Requirements	
Data confidentiality and integrity is paramount Fault tolerance is less important – momentary downtime is not a major risk Major risk impact is delay of business operations	Human safety is paramount, followed by protection of the process Fault tolerance is essential, even momentary downtime may not be acceptable Major risk impacts are regulatory non- compliance, environmental impacts, loss of life, equipment, or production
Architecture Security Focus	
Primary focus is protecting the IT assets, and the information stored on or transmitted among these assets. Central server may require more protection	Primary goal is to protect edge clients (e.g., field devices such as process controllers) Protection of central server is also important
Unintended Consequences	
Security solutions are designed around typical IT systems	Security tools must be tested (e.g., off-line on a comparable ICS) to ensure that they do not compromise normal ICS operation
Time-Critical Interaction	

Less critical emergency interaction	Response to human and other emergency interaction is critical
Tightly restricted access control can be implemented to the degree necessary for security	Access to ICS should be strictly controlled, but should not hamper or interfere with human-machine interaction
System Operation	
Systems are designed for use with typical operating systems Upgrades are straightforward with the availability of automated deployment tools	Differing and possibly proprietary operating systems, often without security capabilities built in software changes must be carefully made, usually by software vendors, because of the specialized control algorithms and perhaps modified hardware and software involved
Resource Constraints	
Systems are specified with enough resources to support the addition of third-party applications such as security solutions	Systems are designed to support the intended industrial process and may not have enough memory and computing resources to support the addition of security capabilities
Communications	
Standard communications protocols	Many proprietary and standard communication protocols
Primarily wired networks with some localized wireless capabilities	Several types of communications media used including dedicated wire and wireless (radio and satellite)
Typical IT networking practices	Networks are complex and sometimes require the expertise of control engineers
Change Management	

Software changes are applied in a timely fashion in the presence of good security policy and procedures. The procedures are often automated.	Software changes must be thoroughly tested and deployed incrementally throughout a system to ensure that the integrity of the control system is maintained. ICS outages often must be planned and scheduled days/weeks in advance. ICS may use OSs that are no longer supported
Managed Support	
Allow for diversified support styles	Service support is usually via a single vendor
Component Lifetime	
Lifetime on the order of 3-5 years	Lifetime on the order of 15-20 years
Access to Components	
Components are usually local and easy to access	Components can be isolated, remote, and require extensive physical effort to gain access to them

2.3 INTERNET OF THINGS

This section presents a brief description of the Internet of Things focused on the technological and security aspects relevant to this work.

The expression Internet of Things (IoT) indicates a class of cyber physical systems that is drawing attention from both the industrial and academic worlds. The interest derives from the employment capabilities in industrial and production sectors, smart home, surveillance, Industry 4.0, and so on. The number of interesting and profitable business cases is large and several companies has already started investing in this area. Cisco estimated the number of connected objects by to be about 50 billions of objects by 2020 [43].

The high-level architecture of IoT can be divided in three layers [79]: sensors, network, and application.

The sensor (or perception) layer, composed by the sensors and actuators that interact with the physical environment, is aimed at collecting and processing measurements to be transmitted to the network

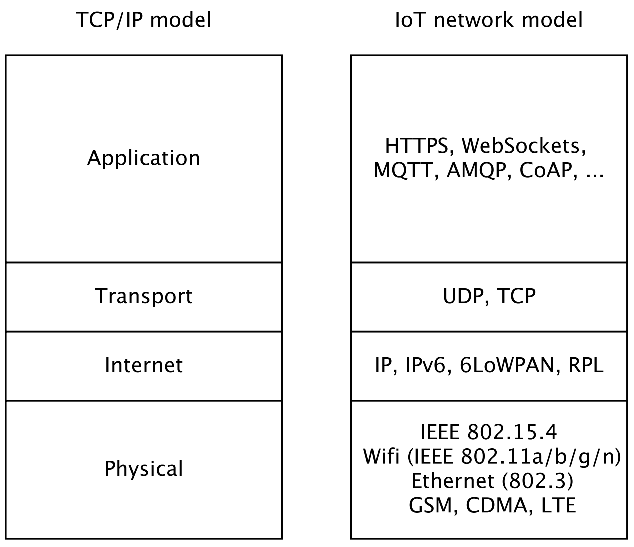


Figure 2.3: IoT network layer model

layer and at operating the actuators. It is similar to the field layer of industrial control systems, but the devices are different. While ICS usually employees PLCs and RTU wired to the actual sensors and actuators, the variety of the so called smart-devices blurs such differences.

The network is used to transmit the information between the sensor and the application layer. It determines the data routes to IoT hubs, devices, and applications. This is probably the most important layer in the IoT architecture because enables the flexibility and compositionality typical of the IoT. The network layer is very different from the process control network of ICS. Figure 2.3 show how example combination of the network layer with respect to the standard TCP/IP model. The main difference lies in the application-level protocol and the way information is routed. ICS typically use fixed topology and fixed communication protocols for the whole lifetime of the installations, data routing relies on the standard layer-3 routing (i.e. PLC are addressed using their IPs), and communication protocols usually do not provide security measures. IoT architectures are often based on publish/subscribe schemas through message brokers, as typical of protocols such as MQTT [2], XMPP [96], AMQP [64], but also use standard HTTPS or two-ways WebSockets. In this way, IoT architectures are flexible and communication protocols provides authentication/authorisation and encryption by design, typically using TLS tunnel of HTTP-based API.

The application layer, not to be confused with the application-level of the network model in Figure 2.3, receives the data from network layer to provide the intended services. In this layer information are stored, analysed, visualised, and processed for any kind of application from smart automation, sport tracking, industrial application (e.g. Industry 4.0), and so on.

2.3.1 *Security of IoT*

Despite the technology heritage of the Internet of Things is not as legacy and critical as the one of standard ICS, still security threats arise from the huge number of expected connected devices.

Researchers has started tackling security challenges related to IoT such as key management issues [114], confidentiality, integrity, privacy, policy enforcements [110,113] among many other challenges. The main works in the literature tried to adapt the security solutions proposed for wireless sensor networks (WSNs) and Internet in the context of IoT. However, we must point out that IoT's challenges take a new dimension which is far from being easy to overcome with traditional solutions. In addition, we must emphasize that most security approaches rely to centralized architectures, making their applications in IoT much more complicated regarding the large number of objects. So, distributed approaches are required to deal with security issues in IoT. In this paper, we survey the different solutions according to two perspectives, namely the security approaches based on traditional cryptographic approaches and the other approaches based on new emerging technologies such as SDN and Blockchain.

IoT systems present different cyber security risks and vulnerabilities at different layers. From our perspective of security monitoring, typical security challenges for IoT can help determining the features of interest for specifying critical condition. For this reason, a short summary is hereafter presented [71].

As the sensor layer is mainly aimed at collecting data, the majority of security attacks focus on forging collected data and destroying perception devices:

- Physical sabotage: the attacker controls the device in IoT by physically replacing it or modifying it [114]. The adversary can also obtain important information from the device, e.g. credentials or secrets, useful for man in the middle attacks, reply attacks, unauthorised data access, etc.

- Injection of malicious code: besides physical attacks, the attacker can run unauthorised code on the leveraging device vulnerabilities, e.g. via unsecure firmware update.
- False data injection: the adversary can replace real data an measurements with forged values, resulting in false information transmitted to IoT applications [109].
- Side channel attacks: the attacker can leverage physical characteristics, like changes in power consumption or electromagnetic signals emitted by circuits, to obtain valuable secret informations.
- Eavesdropping and interference: since many IoT uses wireless technologies, like wifi and ZigBee, an attacker can physically interfere to compromise the network or can use authorisation vulnerabilities to eavesdrop data
- Sleep prevention: many IoT devices regularly switch into sleep mode to save battery. If the attacker prevents this behaviour, the device can became unavailable due to battery drain.

From our perspective, IoT share the same risk of attacks with industrial control system. On the cyber layer, however, IoT presents much more flexible and modern architectures that allows authorised entities to easily retrieve a large amount of information. From this point of view, they look like very suitable for our framework, as our feasibility test shows. Indeed, authorised users can collect command messages flowing in the IoT architectures without compromising other security requirements such as confidentiality, authentication, and authorisation.

3 SITUATIONAL AWARENESS

3.1 OVERVIEW

This chapter presents the so-called Situational Awareness, an high level expression which indicates a set security standards, risk management frameworks, guidelines, software tools, and academic publications related to the continuous monitoring of the security posture of a system at runtime.

Section 3.2 briefly gives an overview of this subject from a risk management point of view. One of our contribution is a through literature review, which is summarised in Section 3.3.

3.2 CONTINUOUS MONITORING

This section explains some key points about continuous monitoring as an effective security solution, from the NIST white paper [98].

The NIST Risk Management Framework (RMF) emphasises the importance of near real-time risk management and continuous information systems authorisation through strong and effective continuous monitoring processes. It also encourages the use of automation to give top-level management the critical information needed to make cost-effective, risk-based decisions that support their primary missions as well as their business processes.

Continuous monitoring applies to many of the RMF's six sequential steps for integrating information security and risk management processes to the NIST risk management hierarchy. Continuous monitoring directly or indirectly supports all six controls:

- Categorising information systems
- Selecting security controls
- Implementing security controls
- Assessing security controls
- Authorising information systems
- Monitoring security controls

Monitoring directly assists in the first step of this process — categorising information systems — from which organisations can derive the secondary benefit of selecting and implementing the proper security controls. Monitoring tools start their processes with initial discovery, usually through passive listening, to determine, among other things, what devices and applications are on the network and the type of traffic, data, and user access with which they're associated. This information helps organisations provide a baseline assessment to determine where they'll need to monitor.

Continuous monitoring enables information security professionals and others to see a continuous stream of near real-time snapshots of the state of risk to their security, data, the network, end points, and even cloud devices and applications. Assessing security controls as well as ongoing monitoring of security controls are both directly assisted by continuous monitoring through vulnerability monitoring processes, which many organisations already have in place.

Many systems, datasets, end points and applications are already being monitoring by system and security administrators with a variety of tools that can be leveraged in the continuous monitoring ecosystem. The most important tools can measure the vulnerability and compliance state of network devices and the security tools themselves, as well as integrate with other security toolsets. This gives organisations near real-time visibility into their compliance state and also aids in incident detection and response by providing additional information to the event management system, often a Security Information Event Management (SIEM) system or log management system.

In this way, continuous monitoring, when implemented through a log manager or SIEM for log and event collection and correlation, helps organisations separate real events from nonimpact events, as well as locate and contain events. Using continuous monitoring for event detection and for vulnerability detection is also becoming part of external discussions about what continuous monitoring includes. For example, the Consensus Audit Guidelines' *Top 20 Security Controls for Effective Cyber Defense cites continuous monitoring of audit logs, continuous assessment, inventory and monitoring for secure configuration as top controls*. The guidelines, developed by government defense and law enforcement agencies in conjunction with the SANS Institute, also list real-time monitoring of account activity, sensitive data movement, malware and threats as equally important components of the continuous monitoring process. This level of integration between vulnerability and event monitoring is critical, given that government entities such as defense agencies are top targets for attackers and hactivists, according to multiple reports.

Because the network is constantly being evaluated, continuous monitoring also greatly improves the level of situational awareness for IT managers. Situational awareness is a term coined by Mica Endsley, who describes the term as having a perception of elements in the environment, understanding the meaning of the elements in the environment, and applying the understanding to being able to project future states. In other words, situational awareness is the awareness of current elements in the monitored environment that are relevant because they may potentially impact that environment today or in the future.

Situational awareness through full network visibility is a key means for mitigating risk. In testimony about real risk reduction to come about through continuous monitoring, the State Department reports a 90 percent improvement in its risk posture after implementing a continuous monitoring program.

Finally, if implemented early in the System Development Lifecycle (SDLC), continuous monitoring can reduce the costs involved with system and application maintenance. Cost improvements are also inevitable when monitoring is both continuous and as automated as possible, including the required reporting documents that support audits.

3.3 RELATED WORK

One of the main source of vulnerability for CPS is the lack of security mechanisms in communication protocols, like authentication, authorisation, and confidentiality [42, 61]. Literature presents several secured version of control protocol, e.g. [46, 50, 80]. However, these security approaches rely on the possibility to redesign and replace at least some parts of the system, while for many industrial control systems downtimes and change management are not practical or affordable due to the high costs and risks related to any possible change. For this reason, redesign is often not an option and legacy components are often present. Passive and unobtrusive security measures are crucial for such CPS.

Intrusion Detection Systems (IDS) have been widely used in ICT security with good results. Signature-based IDS, like Snort [23, 94], are able to express and detect illicit IP packets. Since cyber attacks are combinations of different licit-like actions and communications, signature-based IDS usually fall short in detecting complex attacks.

The *Anomaly-based* intrusion detection approach has proved effective for CPS cyber security [12, 24, 57, 81, 108, 115]. [49] classifies anomaly-based IDS in two main categories:

1. *unattended techniques*, leveraging statistical models or machine learning to create a baseline representing licit behaviours that are compared with the run-time observations

2. *specification-based techniques*, for which a human CPS expert precisely defines what is licit or anomalous in a specification language, and the detection tool compares the state of the monitored system against such specifications.

The absence of human effort is a good advantage of the unattended techniques, but they suffer from high false positive rates which requires human effort to discard false alarms. Our work focuses on the specification-based approach, with the advantage that false positive rates are extremely low or even zero when enough knowledge of the system is available. The main drawback is the effort required to define the known critical conditions. However, CPS typically shows predictable and repeatable behaviours over time. Moreover, the design phase of a critical infrastructure is detailed and documented, providing valuable knowledge to be modelled. Nonetheless, some approaches to automatically derive specifications from the monitored system have proved effective, e.g. [22, 54]. For this reason, specification-based techniques seem to be a good approach for developing security monitors for CPS.

Security monitoring has gained relevance in the Security Operation Centres (SOC) of big organizations and in the DevOps sector. Wide spread frameworks includes Splunk [16, 37], Elasticsearch-Logstash-Kibana (ELK) [51, 53, 95, 105], Grafana [52], and LogRythm [76]. Such tools continuously collect log events and time series data (e.g. cpu load, memory consumption, etc.). Security operators can customise visualisation dashboards of such information to spot anomalous vs. normal behaviours in a graphical way. Moreover, they can also define custom alarms specifying queries on the collected data and events, for instance to detect known indicator of compromise (IoC). The possibility to define alarms is somehow similar to our notion of critical condition described in this work. Unlike our proposed framework, such tools allow queries only on observable data and do not offer a notion of proximity / proximity range from criticality.

SIGNATURE-BASED INTRUSION DETECTION. Signature-based IDS extracts static features from single network packets and verifies whether they belong to a specified set of allowed signatures. They are largely employed in standard ICT networks, since they are easy to use and very effective against the (restricted) class of attacks they address. Snort [94] and Suricata [103] are well known signature-based IDS.

Several works applied this kind of IDS to CPS and in particular to ICS. Cheung et al. [25] propose a model-based intrusion detection technique, which extracts models of the expected/acceptable system behaviours from protocol specifications and detects the ones not conforming with these models. The models are implemented as Snort rules. It leverages the common characteristic of CPS of showing (almost) static topologies, regular traffic patterns, and a limited number of applications and protocols running on the. The technique is based on the TCP/IP field bus protocol (e.g. Modbus/TCP) and derives a protocol specification model for the legal values of packet fields and the their legal relationships. It also constructs normal communication patterns based on the security requirements, the communication directions, and the protocol ports. However, the technique yields a high false alarm rate since it may consider new normal behaviours as anomalous.

In 2008 Digital Bond started Quickdraw, a Department of Homeland Security (DHS) funded research project to "create a passive security event log generator application for legacy field devices that lack security logging capabilities" [13]. The project extends the Snort IDS with additional preprocessors, plugins, and rules specific to ICS. The ruleset is still available for both Snort and Suricata in [14] and [13].

Morris et al. [86] propose an intrusion detection technique for Modbus based on Snort [23]. They describe four classes of intrusion vulnerabilities (denial of service, command injection, response injection, and system reconnaissance) and define Snort rules for detecting them. The detection accuracy relies on the coverage of the Snort rules, i.e. on their combination of quality and quantity. This approach is further improved in [85], which presents fifty signature rules derived from a vulnerability analysis of the MODBUS protocol, and in [47], which describes a set of 28 cyber attacks against industrial control systems which use the MODBUS. Both works substantially improves the detection accuracy.

In our opinion, these are perfect examples of the power and the limitation of signature-based approaches: they can be perfectly employed in CPS whenever the security analysis enables an expert to find specific vulnerabilities or illicit network behaviours, provided that such information is expressible in the signature-based IDS language; on the other hand, more complex behaviour that cannot be expressed as IDS rules are left undetected and the accuracy completely depends on the number and the precision of the rules. In our perspective, network signature extraction, similarly to deep packet inspection, can be a valuable source of observations for our framework, especially thanks to an increasing number of ICS rulesets available in literature, but cannot be considered a satisfactory monitoring technique by its own. This opinion is also supported by a trend in the continuous monitoring best practices for standard ICT that enables established tools [16, 95] to analyses data from parsed network traffic.

The University of Berkeley developed Bro [92], a real-time intrusion detection system which passively monitors a network link over which the intruder's traffic transits. Bro is a programmable high-speed monitoring with real-time notifications and separation between mechanism and policy. It captures network packets and extracts higher level events according to their contents through an extensible protocol parser. Then, it analyses such events based on policy scripts to detect intrusions. Bro provides more flexibility than previous IDS (e.g. Snort and Suricata) that can be valuable in CPS-specific scenarios. Lin et al. [69] extend Bro with a packet parser supporting the DNP3 industrial protocol, analyse the legal values of the different fields in a packet, and develop security policies that match the protocol.

AD-HOC SPECIFICATION-BASED APPROACHES. Hong et al [59] analyse smart grid substations and detect anomalies or malicious behaviours in multicast messages based on the IEC 61850 standard (e.g. Generic Object Oriented Substation Event (GOOSE) and Sample Value technology (SV)), which was issued by the IEC in 2004. The approach detects anomalies and intrusions that violate predefined security rules using a specification-based algorithm. The rule and the logic are very simple and very specific to the vulnerabilities described in the paper, which belong to the Packet Tampering, Replay Attacks and Denial of Service categories. Experimental results of their C/C++ prototype show the feasibility to the extent of the identified attacks a low fault negative rate. This is a good example how specification-based approach can achieve good results

Yang et al. [110, 111] present an intrusion detection system tailored to the IEC 60870-5-104 (also known as IEC 104) network protocol which uses deep packet inspection and combines signature-based and model-based techniques for better accuracy. First, it detects attacks that exhibit well known signatures of the IEC 104. Then, the packets are compared with specified protocol-based and traffic-pattern-based models to detect the anomalous ones. The combined tecnique is implemented as C/C++ modules of the Internet Traffic and Content Analysis (ITACA) tool [62] in the former paper and on top of Snort on the latter. Experimental results prove that such combined approaches can improve the detection accuracy and efficiency of IDS for CPS.

Moreover, the analysis of the protocol can be combined with the analysis of the traffic. Based on communication patterns stipulated in ICS protocol specifications and specific business logics, the detection rules can be extracted and then handed over to the traffic analysis module to improve the accuracy of intrusion detection. Hadeli et al. [54] propose an intrusion detection technique for power systems. It translate configuration files of the control system to configuration files of security tools. More precisely, the approach uses the Substation Configuration Description file defined in the IEC 61850 standard and the Setup Package File of the ABB System 800xA as a formal model of the licit network traffic patterns. Then, it translate such models to actual Snort rules and Firewall configuration using the protocol specifications.

Yusheng et al. [112] proposed an algorithm called SD-IDS (Stereo Depth IDS) performing deep packet inspection for Modbus TCP traffic in real time. It is another approach that combines signature-based with model-based techniques. The algorithm consists of two modules: rule extraction and deep inspection. The former analyses the characteristics of the industrial network traffic and extracts semantic relationships among fields in the Modbus TCP protocol. The latter detect anomalies or intrusions based on the extracted relationships and the real-time traffic data.

TRAFFIC MINING-BASED APPROACHES. Protocol analysis-based approaches generally suffer from a scarce ability to detect unknown attacks, together with performance issues due to deep packet inspection and analysis. Anomaly-based traffic analysis is a possible improvement. The predictability of CPS enables traffic mining to derive useful knowledge for intrusion detection. Traffic mining–based IDS mainly collect traffic data from different networks and subnetworks within a CPS and then apply data mining, machine learning, or statistical data analysis to identify anomalous behaviours.

The approach presented in [101] extracts protocol fields like source and destination IP addresses and ports, the traffic duration, and the average time between consecutive packets from the captured traffic. Then, the traffic data mining is applied to discriminate abnormal behaviours of the system. The approach detects intrusions such as Replay, Denial of Service, Man-in-the-Middle, and Packet Tampering.

The technique in [60] is based on the probabilistic Principal Component Analysis (PCA) applied to network traffic in industrial networks. The first result is that random burst traffic is a source of false alarms. Then, the IDS builds a probabilistic model for the traffic matrix and evaluates the impact of random bursts on that model. The technique discriminates abnormal traffic of ICS measuring the change of the rank. Their experimental results prove this method able to mitigate the interference of random burst traffic to intrusion detection.

Previous example shows probabilistic and statistical approaches applied to traffic analysis in CPS. Machine learning is another set of techniques that proved effective in the same area. Its ability to analyse large amounts of data can be leveraged to detect unknown intrusions on control systems. Neural Networks are able to capture nonlinear relationships between traffic features and their security states. Traffic data-based training is used to build a model of normal behaviours. Real-time data is then classified using such models to detect abnormal traffic. The work presented in [106] extracts network traffic features like packet size, ICMP protocol ID, sequence numbers, and IP protocol fields to build input vectors for neural network training. The resulting classification detects attacks like DoS and eavesdropping. These results are improved in [73] using a technique employing a moving time window where packet streams are considered as time series. As a result, a better set of traffic patters and features is extracted for the collected data. Moreover, the combination of Error-Back Propagation and Levenberg-Marquardt neural network learning algorithms further improves detection accuracy.

Many machine learning-based anomaly detection for cyber physical system appear in literature, proving the effectiveness of such approaches on one hand, but also the need for improvements on the other, for instance to mitigate the requirement of high volumes of unlabelled or labelled training data and to lower the high false positives rates. Most of the related works propose a vast spectrum of fine-tunings and combinations of machine learning technique to improve the accuracy of the detection like fuzzy logic classifiers [72, 74, 75], Support Vector Machine (SVM) [77, 78], Ant Colony Clustering Model (ACCM) [104], ad-hoc clustering-based approaches that leverage a deep understanding of the CPS [66, 67], information entropy–based method for traffic feature extraction plus discrete cosine transform (DCT) and singular value decomposition (SVD) to create digests of normal/abnormal behaviours [40].

Caselli et al. [20, 21] present an intrusion detection targeted at systems that use the IEC-104 network protocol leveraging deep packet inspection. It observes a selection of packet fields and builds a Discrete Time Markov Chain (DTMC) of the traffic patterns, where the states represent the information of the observed control message, while a transition from state s_1 to s_2 represent that s_2 is the next message that follows chronologically the message that s_1 represents. Probabilities attached to transitions are estimated using the frequency of the transition within the observed data. Sequence attacks are simply sequences of control messages. After building the Markov chain, the technique compares the observed behaviour with the model detecting state violations (i.e. unexpected control messages), transition violations (i.e. the observation of a message that should not follow the previous one), and transition anomalies (i.e. sequence of messages that do not adhere with the expected probability). The approach is further improved in [45] that build much smaller models retaining comparable accuracy.

The main results of intrusion detection based on traffic analysis is that the accuracy of the vast spectrum of possible techniques for CPS may depend on: (i) the actual protocol packet fields extracted and collected for the observed network traffic: this requires a good understanding of the CPS (ii) the actual traffic patterns and feature of the control messages: to this aim both CPS-agnostic features (i.e.

connection topology and message directions, bandwidth, and time between messages) and CPS-specific ones (i.e. the expected sequence of messages for a correct control of the system) must be considered for higher accuracy (iii) the right combination and fine-tuning of the statistical or machine learning techniques used.

In all these cases, anomalies are observations that diverge from the typical ones. Since anomalous does not necessarily means illicit, all such techniques suffer from high false positive rates. While the main advantage of these methods lies on the small human effort for training the model and detecting anomalies, in practical cases these techniques still need human effort due to both the large number of false positive alerts and the fine-tuning required for better accuracy.

CONTROL PROCESS ANALYSIS. Previous works focus on intrusion detection techniques for protocol and/or traffic analyses that use the specificities of the cyber components of CPS. Another category of works leverages the full semantics information of both the cyber and the physical layer of the monitored CPS.

The approach in [68] focuses on attacks that ultimately impact the physical process with undesired effects. The work presents a process-aware technique that detects whether a sensor signal is corrupted, ensuring sensor data veracity in a CPS. The approach uses a set of different real-time algorithms for spoofing sensor signals and the physical laws behind the process. This appears complementary to our framework, which instead assumes the integrity of the observed valued. Thus, [68] could be be integrated in our approach: the alerts from the different algorithms could be treated as observations in our framework and used as variables to express better security critical conditions.

Colbert et al. [27] present two control process–oriented detection methods for intrusion detection in control systems. Unlike traditional anomaly-based IDS, the techniques require process engineers and plant operators that deeply knows insights of the process. The first method uses Critical Process Variables and thresholds of their licit values defined by plant operators, who has the best knowledge of the critical assets of the system. Sensors measure real-time values of the variables and send an alerts when such values do not satisfy the defined thresholds. The possible constraints have the form *Process Variable from PLC* $\bowtie$ *value*, where $\bowtie$ can be greater than, less than, equals, not equals, and change (by a delta value). The second method uses cyber metrics that are identified again with the help of plant operators. It is worth noticing that this approach is a simpler proper subset of less recent papers [17, 29, 89] of which present thesis is an improvement in all respects. As a consequence, [27] further validates the feasibility of our approach within critical infrastructures. In particular, it proves how the knowledge of the CPS can be greatly beneficial to monitoring and detection of cyber attacks, which is the main assumption of our work.

Nai et al. [18, 88, 90] developed a specification-based Intrusion Detection and Prevention System methodology specific for SCADA systems that is not based on specific attack models and can detect 0-day attacks. The methodology allows combining the knowledge of the physical process with the cyber behaviour to monitor, and is further extended in [30] with a greater expressiveness and more effective computation methods.

Kiss et al. [66] present a method to detect cyber attacks targeting measurements sent to PLCs. It uses the Gaussian mixture model to cluster sensor measurement values and a cluster assessment technique known as silhouette. The authors prove the feasibility of the approach and show results that outperform the k-means clustering algorithm within a simulation testbed of a chemical process and three different cyber attacks.

Gao et al. [48] develop a set of command injection, data injection, and denial of service attacks exploiting the lack of authentication of most SCADA network protocols. Then, they capture network traffic both of normal operations and during such attacks. Collected data are used to train a neural network for intrusion detection, also using the knowledge of the physical process to detect false response injection attacks. The experimental results prove that including features from the physical layer yields higher accuracy rates.

The approach presented in [87] focuses on attackers that forge measurement signals in the control loop to influence control decisions. The authors present a detection system targeted to power grid that uses a semantic analysis framework based on the physical model of the system, proving how the knowledge of the process add crucial information to detect cyber attacks.

Since some attacks interfere or change control commands, a class of detection techniques focuses on the analysis of control commands. Carcano et al. [19] define a specification language for critical states and propose an approach that compare information collected through deep packet inspection of the Modbus protocol with the specifications. Features from the control commands occur in critical state specifications. The logic language is very simple and only enables expressing thresholds on the value of the extracted features and variable, extremely similar to the aforementioned and more recent [27]. The approach assumes that it is always possible to extract and observe the variables occurring in the specifications.

Lin et al. [70] present a detection system aimed at power grids and DNP3 control protocol. They define a technique that combine knowledge of the cyber and of the physical layers, to respectively check the integrity of the control commands and to compare extracted features and measurements with a semantic model of the system. The prototype is based on Bro and the semantic is based on threshold verifications. This approach shows how capturing a combination of cyber and process features is crucial to identify express critical specifications. However, the expressibility of the language is strongly limited to constant threshold verification.

MODEL-BASED APPROACHES AND FORMAL METHODS. Model-based approaches aim at constructing a model of the system that captures the main characteristics of its behaviour. Once a model is developed, it is possible to specify formal properties and to use model checking to prove that the modelled systems verifies such properties.

For this class of problems a large literature exists aimed to synthesise system models [56], to ensure that verification results about models apply to the CPS implementations [82], to find and verify invariant properties of parameters of interest [26], with large variety of formal methods and verification techniques.

The main disadvantage of model-based verification is the effort required to develop a formal method which is correct and accurate. Fortunately, part of the formalisms and results from model checking can be adapted to online monitoring (e.g. [6]), which can be defined as the verification that the known prefix of the traces (or signals) that the system exhibits has not violate the specified properties.

Most online monitoring techniques are based on temporal logics [6, 10, 36, 39, 44, 58], which are an established way to specify temporal properties based on the set of atomic propositions that are satisfied by the states of the system.

Several variants of temporal logic exists. In particular, at least two appears as the most common in works for runtime verification of the cyber physical system: the Metric Temporal Logic (MTL) [3, 41, 44] and the Signal Temporal Logic (STL) [36]. Each of these can have variants, e.g. in terms of future vs. past modal operators, or in terms of subclasses with different verification complexity.

Several works employ Satisfiability Modulo Theories (SMT) applied to CPS. [100] estimates the state of a time-invariant physical discrete-time developing a SMT-based Luenberger observer for system whose sensors are manipulated by an attacker. Some works reduce temporal logics to SMT problems. [35] develop an SMT-based monitoring prototype for LTL using the Z3 solver [33]. [113] reduces LTL properties to SMT and checks it against models of embedded systems. [10] reduces the satisfiability of constraints LTL over clocks to a decidable SMT problem, obtaining a complete Bounded Satisfiability Checking procedure implemented by using standard SMT solvers.

SUMMARY. The extended literature review shows that there are two main kinds of online security monitoring approaches applied to CPS. The first is the intrusion detection approach, which focuses on extracting and analysing valuable features from the CPS and that only alerts if and when a possible attack or anomaly is detected. Such works employ several feature extraction methods, ranging on standard ICT metrics, deep packet inspection of specific control protocols, physical control parameters, wireless sensor networks, and power consumption or similar low level measurements for detecting side attacks. They also present different analysis techniques, ranging on statistics, machine learning, and physical models to compare observed values with the expected ones.

From our perspective, intrusion and attack detection approaches are either *unattended*, i.e. the initialisation phase does not require human effort, and *specification-based*, i.e. security and/or process engineers are required to provide their expertise to specify illicit or unwanted conditions. The former category appears much more mature, and from working experience there are vendors on the market that provide effective appliances and services based on unsupervised or semi-supervised machine learning techniques specific to CPS. The latter category has the advantage of much lower or nearly null false positive rates but appears much less mature.

The other kind of approach is runtime verification, which focuses on formal properties specifications and on possible algorithms that verify at runtime if the system has not violated (yet) the specifications. Formal methods are widely used in this area and present very interesting results. Most works abstract from the actual features of the monitored system, while the specification language and the monitoring algorithms usually capture specific needs of illicit behaviours of CPS, like the signal or the metric temporal logics.

From my working experience, there exists a practical perspective to continuous monitoring derived from DevOps, based on the deployment of SIEMs, timeseries databases, and visualisation consoles. In this way, the console constantly shows a quantitative picture of how good or bad the system is behaving, unlike intrusion detection and runtime verification that stay quiet until an alleged attack or violation is detected. Moreover, operators can refine dashboards and perform drill-down queries on the history of the observations timeseries database as a support of common practices employed in Security Operation Centres (SOC) and Computer Security Incident Response Teams (CSIRT).

The aim of this work is to combine a real-time quantitative notion of criticality with specification-based detection of violations. We develop a proximity-based notion of quantitative criticality of a state with respect to a critical specification, and this appears as a novelty of our proposed framework.

Moreover, most works in the literature assume that the features of interest are always observable or equivalently that the verification of atomic proposition is alway possible. Some notable exceptions exist, but focus on different ways to evaluate the missing values with probability distribution, model-based simulations, control theoretic arguments, and other methods. Our approach is aimed at handling unobservable variables and reasoning on missing values without trying to evaluate, guess, or predict them, for a greater applicability to real cases. The possibility to handle and reason about unobservables, and to provide further knowledge refinement from human operators to the reasoning engine, appears as a novelty of our framework.

Part II

THE MONITORING FRAMEWORK

4 MOTIVATING EXAMPLES

4.1 OVERVIEW

This chapter presents two realistic motivating examples to intuitively explain our monitoring approach. We also developed a working simulation testbed of each example to test and validate the feasibility of the prototype of our framework.

The example are supposed to clarify the meaning of the variables of interest to be observed, the specification of critical condition on top of variables, example of unobservable variables necessary for a critical condition, and critical specifications based on observation times.

Section 4.2 describes an automation system for building heating control. Section 4.3 presents an industrial chemical process with more insights about unobservable control application internal states and simple observation time specifications.

4.2 BUILDING HEATING SYSTEM

This section presents an example of CPS which is overly simplified but still capable of explaining the kind of anomalies our framework is able to detect, its capability of handling unobservable aspects, and its notion of predictiveness. The following chapters also refer to this example to ease the explanation.

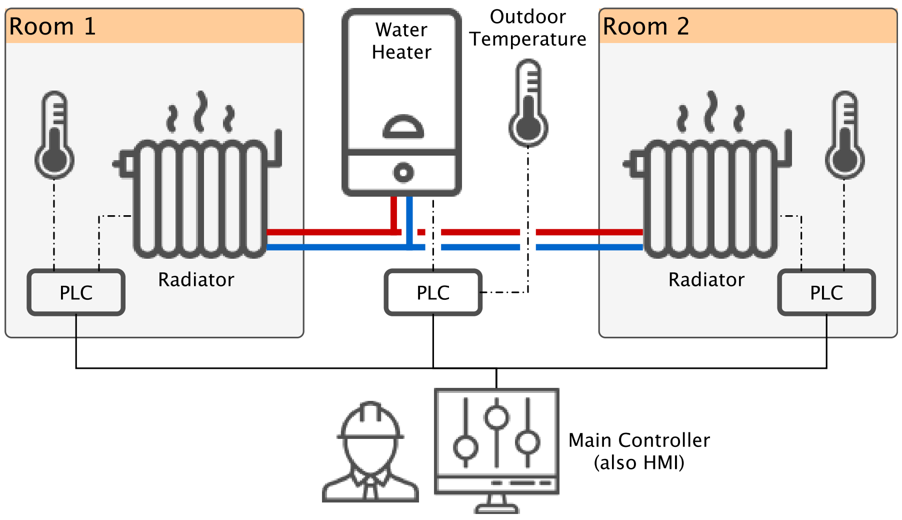


Figure 4.1: Two rooms building automation example.

Figure 4.1 shows a simplified building automation system controlling the temperature of two rooms. The CPS is made of the following components ($i = 1, 2$):

1. a thermometer in each room that measures the temperature T_i
2. an external thermometer for the outdoor temperature O
3. a radiator R_i in each room that can be switched on/off
4. a main water heater H that can be switched on/off.

Each room i has a setpoint S_i representing the desired target temperature of that room. Sensors (the thermometers) and actuators (the radiators and the heater) are wired to Programmable Logic Controllers (PLC). Each PLC is connected to the same TCP/IP-based Process Control Network (PCN). The Main Controller (MC), connected to the PCN, is able to send read and write commands to the PLCs. In this example we assume the Modbus is used [61, 84], a very widely used control protocol with no security mechanisms for authentication/authorisation.

In this example the main controller provides a Human Machine Interface (HMI) that allows an operator to visualise the current process parameters and to manually operate the system. An operator uses the HMI component of MC to: check temperatures and the on/off status of the radiators and of the heater; set the desired temperature of the rooms (setpoints); turn on/off the radiators and the heater. Besides manual human operations, our example CPS is automatically oper-

every 500 ms	$\rightarrow$	read T_i, S_i, R_i, H, O
$T_i < S_i$ and $O < S_i$	$\rightarrow$	write $R_i =$ **true** and $H =$ **true**
$T_i > S_i$	$\rightarrow$	write $R_i =$ **false**
$T_1 > S_1$ and $T_2 > S_2$	$\rightarrow$	write $H =$ **false**

Table 4.1: Example of automatic operation rules.

ated through a set of rules implemented in the main controller MC and listed in Table 4.1.

In this example the main controller is the only device that is supposed to send read and write Modbus commands to the PLCs. Suppose that an attacker (e.g. a malware) compromises the main controller and sends malicious Modbus commands to the PLCs from it. Such illicit commands would have exactly the same network signature and the same payload as the licit ones. Thus, neither signature-based IDS (e.g. Snort [23, 94], Suricata [103]) nor basic Modbus deep packet inspection tools (e.g. Wireshark [32, 91, 97], Bro [93]) can detect such anomalies. Indeed, the only way to detect them is to understand that such Modbus commands do not conform with the expected behaviour of the system.

Assume that the attacker sends read commands to the PLCs to gather and exfiltrate process information. Read commands and their responses are identical to the licit ones, however a network activity showing an unexpected read frequency can be considered illicit, expressed by the following *critical condition*

$$|F - f_{expected}| > \epsilon \qquad (\phi_1)$$

where F is the number of read commands per seconds *observed* from network capture, $f_{expected} = \frac{3}{0.5}$ corresponds to 3 PLCs and 500 ms from Table 4.1, and ϵ is a tolerance constant.

Similarly, suppose the attacker sends a Modbus write command $H =$ **false** to turn off the water heater to prevent room 1 or room 2 from reaching the desired temperature. Although this command is identical to the licit ones, it might be possible to detect such illicit activity when the presence of the write command does not conform with the expected behaviour from Table 4.1, expressed by the critical condition

$$\neg H \wedge \big((T_1 < S_1 \wedge O < S_1) \vee (T_2 < S_2 \wedge O < S_2)\big) \qquad (\phi_2)$$

Intuitively, critical condition Eq. (ϕ_2) holds if a turn off command is sent when the heater should be on according to Table 4.1.

Notice that the critical condition Eq. (ϕ_1) purely addresses the cyber layer of the CPS, while Eq. (ϕ_2) mixes cyber and process aspects allowing for a greater expressiveness and effectiveness of the approach.

In this example a sensor might stop working, e.g. the outdoor thermometer, and the corresponding value might become unobservable. The novelty of our framework is the capability of handling both *observable* and *non-observable* variables, improving its range of applicability. Moreover, condition Eq. (ϕ_2) might not be critical if a human operator intentionally operates the CPS manually, e.g. for maintenance. A more accurate critical condition can be defined as

$$\neg M \wedge \neg H \wedge \big((T_1 < S_1 \wedge O < S_1) \vee (T_2 < S_2 \wedge O < S_2)\big) \qquad (\phi_3)$$

where the boolean value M represents the manual and intentional operation of the human operator. The assumption that a monitoring tool is aware of human intentions is unreasonable in practical cases. Thus, M must be treated as unobservable, yet is necessary for a better accuracy.

If the example CPS is in a state where the critical condition Eq. (ϕ_1) is not satisfied, a notion of *proximity* from Eq. (ϕ_1) can be defined, representing how far the measured read command frequency is from its expected value. Monitoring how the proximity value changes in time enables to monitor if the CPS is approaching the critical condition Eq. (ϕ_1).

4.3 A SIMPLE CHEMICAL PROCESS

This section presents an example of a simplified chemical process and its control system and logic. Figure 4.2 shows the components of the chemical process.

Process overview. A pharmaceutical company produces a chemical product P from three reagents A, B, and C. All chemical reagents and products are liquids. The process begins filling a reactor with reagents A, B, and C with concentrations of respectively 70%, 20%, and 10% using precise pumps. Assume these proportions are part of a patented secret process. More than one reactor can be used in parallel: this example considers two reactors L and R. When the reactor L (or R) is filled, the chemical reaction takes 30 minutes, after which the content of the reactor is moved to the final product silo S using output pumps *pLS* (or *pRS*). Input pumps pA pB pC fill the tanks containing reagents A B C when the level of the tank is lower than a threshold. Pumps *pAL*, *pBL*, *pCL* and *pAR*, *pBR*, *pCR*, which are used

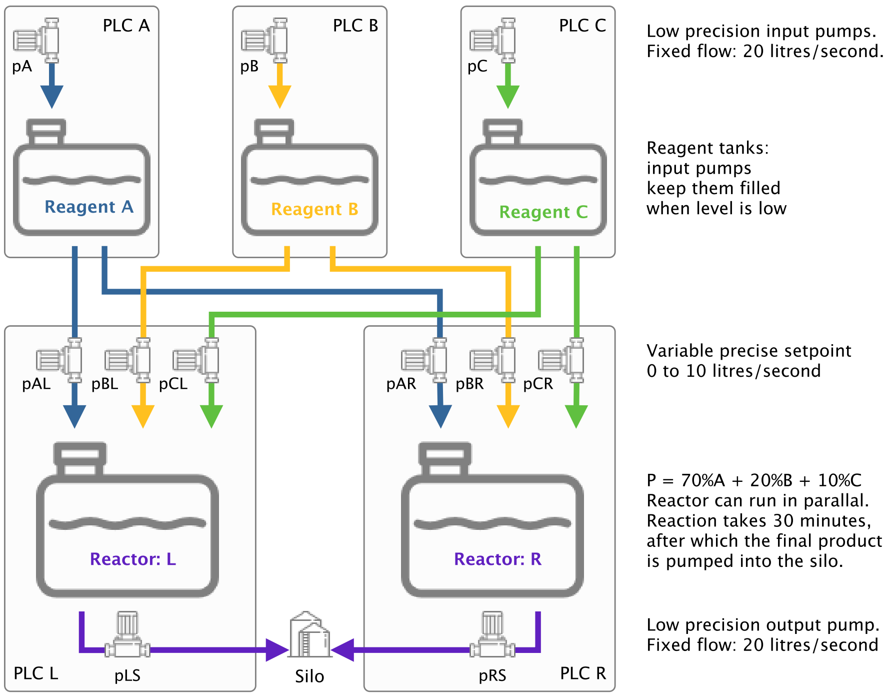


Figure 4.2: Simple chemical process use case.

to mix reagents in the correct proportions, are required to be precise: a numerical setpoint specifies the pump flow (in this example from 0 to 10 litres per second). Pumps *pA*, *pB*, *pC* and output pumps *pLS*, *pRS* do not need to be precise, the pump flow is fixed to 20 litres per second, and they can only be turned on and off.

Cyber components. The control of the process is based on the level sensors: each tank, reactor, and silo have a sensor that measures the *level of the content*. The employed actuators are: mixing pumps *pAL*, *pBL*, *pCL*, *pAR*, *pBR*, *pCR*, which can be operated setting a *variable setpoint* and can be switched *on/off*; other pumps *pA*, *pB*, *pC*, *pLS*, *pRS* which can only be turned on/off.

Sensors and actuators are wired to Programmable Logic Controllers (PLC), connected to the same TCP/IP-based Process Control Network (PCN):

- PLC A, PLC B, and PLC C read level sensors of resp. tanks A, B, C and control the on/off status of input pumps A, B, C.
- PLC L reads the level of reactor L and controls the setpoint of pumps *pAL*, *pBL*, and *pCL* and the on/off status of *pLS*. Analogously for PLC R.

A SCADA server, connected to the PCN, controls the chemical process sending read and write network command to the PLCs through an industrial control protocol like Modbus [84]:

- it constantly reads control parameters from the PLCs using active polling at a constant frequency, specified in its configuration;
- it automatically operates the process implementing a control logic, sending write commands to the PLCs when certain predefined conditions occur;
- it provides a Human Machine Interface (HMI) component, which shows the current values of the process and enables operators to manually send control commands to the PLCs.

The SCADA server is the only system that is allowed to send read and write commands to PLC, as a result of automatic or manual operations.

Specifying criticalities from the knowledge of the process. An attacker may compromise the SCADA to gather and exfiltrate secret process data off the PCN or to damage the process. While any network message not originated from the SCADA can be easily detected as illicit, read and write commands sent from a compromised SCADA are identical to the licit ones from the network signature perspective. Thus, signature-based IDS fail short to detect such attacks. Modbus-like control protocols, vastly used in existing industrial control systems, present no authentication/authorization mechanisms. Hence, the attacker can initiate Modbus TCP connections from the compromised SCADA server to any PLC to illicitly operate the process.

Suppose the attacker sends read commands and collects the response values to exfiltrate secret data, like the reaction proportions and timings. This attack can be detected comparing the total number of read messages with the one expected from the SCADA server configuration. Let RF_p be the number of read commands from the SCADA server to PLC p in the time unit, with $p \in \{A, B, C, L, R\}$. This value can be easily observed using network traffic analysis and deep packet inspection tools like Wireshark [107]. Let rf_p be the constant number of read commands per second to the PLC P in the SCADA server configuration, called polling frequency. The critical condition corresponding to the read attack is then

$$RF_A \neq rf_A \vee RF_B \neq rf_B \vee RF_C \neq rf_C \vee RF_L \neq rf_L \vee RF_R \neq rf_R \quad (4.1)$$

Table 4.2: Example of automatic operation rules.

every 500 ms	$\rightarrow$	read *all* sensors	
if $Level_X < h_X$	$\rightarrow$	set $pX =$ **off**	for $X \in \{A, B, C\}$
if $Level_X > l_X$	$\rightarrow$	set $pX =$ **on**	for $X \in \{A, B, C\}$

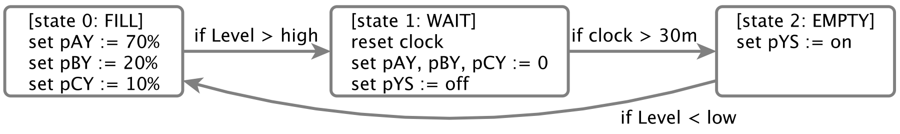


Figure 4.3: Automatic reactor control statechart ($Y \in \{L, R\}$).

In this work RF_p are called *variables* of interest of the monitored CPS. Each variable is bound to an observation method, in this case to network packet inspection that counts read commands.

Suppose the attacker sends write commands with random setpoints to the mixing pumps to alter the proportions and to corrupt the chemical reaction. Let PS_p be the variables representing the last observed setpoint sent to the PLC controlling the pump p. Again, it is easy to observe PS_p with deep packet inspection of Modbus commands on the PCN. It is possible to detect such attack comparing those values with the expected proportions, expressed by the following critical condition:

$$\begin{gathered}(2 \cdot PS_{pAL} \neq 7 \cdot PS_{pBL} \wedge PS_{pBL} \neq 2 \cdot PS_{pCL}) \quad \vee \\ (2 \cdot PS_{pAR} \neq 7 \cdot PS_{pBR} \wedge PS_{pBR} \neq 2 \cdot PS_{pCR})\end{gathered} \tag{4.2}$$

Unobservable variables. The aim of industrial control systems is to automatically operate sensors and actuators to implement a specific process. In our example, automatic control rules are implemented by the SCADA server. Figure 4.3 shows the state-based rules for sensors and actuators that control the reaction, while Table 4.2 shows the rules to control the other components.

Suppose the attacker sends on/off command to the *pLS* pump that does not comply with the control logic. The following critical condition detects such attack:

$$(S_L = 0 \vee S_L = 1) \wedge pLS = \text{on}) \vee (S_L = 2 \wedge pLS = \text{off}) \tag{4.3}$$

where variable *pLS* is the observed on/off payload of the write command and variable S_L is the state of the control logic of reactor L. Notice that S_L is necessary to express critical condition Eq. (4.3), but it is inherently unobservable because it is the hidden state of a control program implemented in the SCADA server. While S_L is always unobservable, any variable may become temporarily unobservable, e.g. when it is bound to a malfunctioning sensor. This example shows that the assumption that all the variables are always observable is not feasible with real cases, and that critical conditions of interest may need to refer to unobservable variables. Next sections show how our framework is capable of dealing with them.

Observation time in critical specifications. Some attacks can be detected observing how the CPS behaves in time. According to Figure 4.3, when a turn-off command and later a turn-on command are sent to pump *pLS* (i.e. a transition from state 1 to 2 occurs), then the two write commands must be observed with at least 30 minutes time difference, but not much more than that. Assuming 1 minute tolerance in the execution of the control logic, the following critical condition detects attacks that turn off *pLS* too early or too late:

$$\neg(pLSon.t < pLSoff.t \to 30m < pLSoff.t - pLSon.t < 31m) \tag{4.4}$$

where *pLSon* and *pLSoff* are boolean variables bound to the observation of respectively on and off write commands to pump *pLS*.

Refinements. Detecting if the current state of the CPS is critical w.r.t. a critical specification may be impossible in presence of unobservable variables. A human operator can provide the monitor with a refinement, i.e. a logical expression of further knowledge. For instance, a process operator who supervises the production knows whether a reaction started, i.e. if $S_L = 1$. For this reason, the operator can alternatively provide our monitor with the refinement $S_L = 1$ or the refinement $S_L = 0 \vee S_L = 2$.

Similarly, unobservable variables can express human intentions, which can be valuable knowledge to a monitoring framework. Assume an operator sends licit commands for maintenance purpose not compliant with the control logic of the CPS. Critical conditions Eq. (4.2) Eq. (4.3) Eq. (4.4) do not discriminate such licit commands from the attacker's ones. Each condition ϕ can be replaced with $\neg M \to \phi$, where M is an unobservable boolean variable representing that the CPS is intentionally manually operated. This way, an operator provides the refinement $M = \textbf{true}$ when the maintenance activity begins and $M = \textbf{false}$ when it ends, and his operations are not detected as illicit.

When it is not possible to discriminate the criticality of the current state of the CPS due to unobservable variables, our monitor is capable of computing an *assisted check*, i.e. a logical expression that a human operator can evaluate in order to provide a minimal valuable refinement. Next sections show how the monitor computes this expression using its logical reasoning core.

Predictiveness. Besides discriminating whether the current state of the CPS is critical, our monitor also predicts if the system is getting closer to a critical condition. To this aim, the monitor computes a notion of distance of the current CPS state from a critical condition. When the current state is non-critical, monitoring how the distance from the critical condition changes in time tells if the system is reaching that criticality. On the other hand, when the state is critical, it is possible to monitor the distance from the border of the criticality, i.e. how far the CPS is from returning to a non-critical state.

In our example, if the current CPS state satisfies the critical condition Eq. (4.1), the criticality measure represents how much the observed polling differs from the expected one. On the other hand, if the current state is not critical w.r.t. Eq. (4.4), i.e. the observation time between on and off commands is between 30 and 31 minutes, the the state is not critical, the criticality is a negative value, and its absolute value represents how the observed values are close to the critical boundaries 30 or 31.

5 THE CORE FRAMEWORK FOR OBSERVABLE CPS

5.1 OVERVIEW

This chapter describes the core of the proposed monitoring framework for CPS, presenting formal definitions, possible computational methods, and their correctness proofs.

As described in Section 3.3, a number of works in the literature shows that it is possible to leverage the predictability of CPS and the domain knowledge of human experts to gather valuable information to develop specific security monitoring and detection solutions. We follow this line and assume that process and security engineers can derive a set of critical conditions that the CPS should not reach.

The framework aims at security monitoring feasible Security Operation Centres (SOC) and Computer Security Incident Response Teams (CSIRT), enabling operators to:

1. define the aspects of the CPS to be observed, called variables, and the related observation methods;

2. specify the critical conditions of interest in terms of observed values;

3. specify a notion of proximity among system states, used to define and compute a quantitative criticality of the CPS from observations;

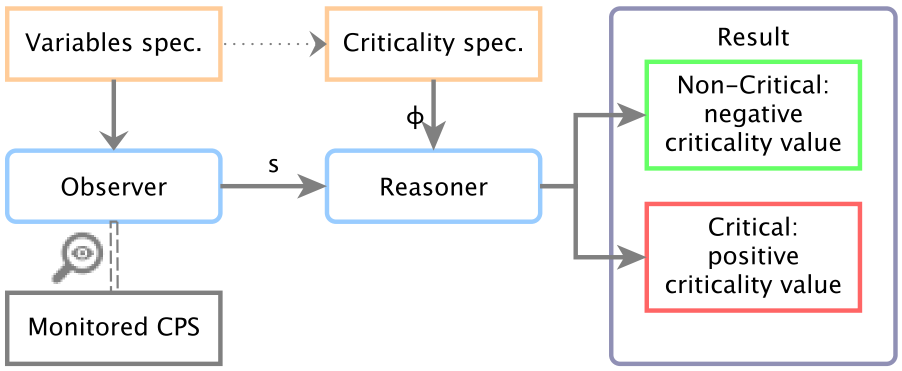

Variables spec.
Criticality spec.
Result
Non-Critical: negative criticality value
ϕ
s
Observer
Reasoner
Critical: positive criticality value
Monitored CPS

Figure 5.1: Structure of the monitoring framework.

4. continuously monitor the changes in time of the criticality as a measure of how closely and quickly the system is evolving towards a critical condition or is returning to a licit state.

Most literature on attack detection for CPS assumes the features of interest are always observables. This chapter presents our proposed framework under the same assumption. However, this might be unfeasible in real cases. In Chapter 6 we relax this assumption and modify the framework to handle unobservables.

Figure 5.1 depicts the main structure of the framework. Given the CPS variables and critical conditions specifications, the monitor runs in parallel with the CPS, continuously observes its current state, and evaluates its criticality with respect to the critical conditions. The main components are the *observer* and the *reasoner*.

The observer is aimed at extracting the features of interest of the CPS. Each feature is associated to a variable. The framework is agnostic about the kind of parameters to measure, however the following sources of information appear particularly relevant according to the literature and the experience from the industry, as described in Section 3.3:

1. Network traffic: passive capturing collects a large set of ICT metrics that characterise the cyber layer of the control devices. Deep packet inspection provides insights about control parameters of the physical process. Since critical plants seldom tolerate active network scanning and agent-based information gathering, the network traffic captured from process control networks appears the main feature extraction source: it provides both standard ICT security metric, like the connection topology and bandwidth, and CPS-specific parameters, like values from sensors and commands to actuators.

2. Application logs and performance metrics from SCADA servers and HMI, as a standard practice in security operation centres. Beside standard ICT system logs, historian applications log the value of process parameters for compliance and troubleshooting purpose which can be used as an alternative observation technique to deep packet inspection.

3. IoT message queues. The network layer of IoT architecture often employs message queues over VPN or TLS tunnels. This prevents from passive network capturing. Fortunately, an authorised entity can connect to the message broker, subscribe to relevant queues, and obtain full visibility retaining confidentiality and authentication requirements.

Our threat model assumes the integrity of the observed values. This is a common assumption in a number of detection approaches based on features extraction and analysis found in the literature. Some observations are difficult to compromise, e.g. the bandwidth consumption measured through the SPAN port of network switches. The number and variety of observed values can mitigate this risk, especially with redundant observations. For instance, queries to historian applications can be compared with data from deep packet inspection of the control network to detect data integrity attacks. Industry 4.0 approaches also suggest deploying additional IoT devices to detect physical sensor tampering. Both examples can be adopted in our framwork associating different variables to different sources of the same data. Model-based approaches [83] can detect integity attacks comparing the observed values with the expected ones. These can be complementary to our framework: the output of a model can be regarded as an additional variable to compare with observed raw data.

The reasoner component iteratively receives the current CPS state s from the observer and evaluates its criticality with respect to a critical condition specification ϕ. Each critical specification consists of:

1. a *critical formula*, i.e. a boolean combination of linear constraints on the variables representing an illicit or unwanted condition of the CPS

2. a notion of *distance* on the set of states, used to compute the criticality of the current state from observations

3. an optional set of *criticality thresholds* which represent different levels of alerting. Thresholds are necessary for the criticality estimation described in Section 5.5.

The reasoner discriminates whether the observed state is critical. In case of fully observable variables, this is simply achieved checking if the values satisfy the critical formula. Then, it evaluates the criticality measure, which represents:

1. The distance to licit states when the state is critical. In this case the criticality is a positive value. If this values decreases in time, the system is evolving to licit states.

2. The proximity to critical states when the status is non-critical. In this case the criticality is a negative value. If this value increases in time, the system is getting closer to the critical condition.

In this way, monitoring how the criticality changes in time permits to predict how the system is evolving with respect to the specified critical conditions.

Section 5.2 describes how to specify the variables of interests and focuses on observations relevant to cyber physical system. Section 5.3 presents the critical condition specifications, defint a quantitative notion of criticality of the observed state of the system, and describes possible techniques to evaluate such criticality.

5.2 SPECIFICATION OF SYSTEM VARIABLES

The specifications of variables allow the observer to extract the features of interest. The specification of a variable consists in:

1. The *name*, used as an identifier in the specification of critical conditions.

2. An optional *range constraint*: it constraints the acceptable values for the variable in the form of upper and lower bounds.

3. The *observation method*: an executable procedure to gather the current value of the variable. Example are the execution of deep packet inspection, log analysis, or the subscription to IoT message queues. Since the framework is aimed at nearly real-time monitoring, the observation method needs to be nearly real-time as well.

Let $\mathcal{V}$ denote the set of variables, i.e. the parameters, aspects, events, measurements of the CPS that are required to express a known critical condition. A variable can be boolean, integer, or real, and $\text{range}(v)$ denote the range constraint of v defined by the lower and upper boundaries in the variable specification. In the following, boolean variables range on the set $\{0,1\}$, with both the boolean and the numeric meaning, in order to be able to use boolean and numeric variables in the same arithmetic expressions. For instance, the expression $b_1 + b_2 + b_3 \leq 2$ denotes that at most two boolean variables among b_1, b_2, b_3 are true. As a consequence, all variables in $\mathcal{V}$ also range on $\mathbb{R}$.

Definition 5.1. Let $V \subseteq \mathcal{V}$ be a subset of variables. A *partial assignment* (or simply an assignment) is a function $a\colon V \to \mathbb{R}$ that maps variables to values such that $a(v) \in \text{range}(v)$ for each $v \in V$. The notation $\text{dom}(a)$ denotes the domain V.

Definition 5.2. A *state* of the monitored CPS is an assignment s such that $\text{dom}(s) = \mathcal{V}$, i.e. a total assignment of variables. The set of all possible system states is denoted by $\mathcal{S}$. Given a partial assignment a, we define

$$\mathcal{S}(a) = \{s \in \mathcal{S} \mid \forall v \in \text{dom}(a)\colon s(v) = a(v)\}$$

as the set of states *consistent* with the assignment a.

5.3 SPECIFICATION OF CRITICAL CONDITIONS

Definition 5.3. A *critical codition* is a logical formula defined by the grammar:

$$\phi ::= a_1 v_1 + \cdots + a_n v_n \bowtie b \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi$$

where $v_i \in \mathcal{V}$, $a_i, b \in \mathbb{R}$, $\bowtie \in \{<, \leq, >, \geq, =, \neq\}$. The set of variables occurring in a formula ϕ is denoted by $\text{var}(\phi)$ defined as

$$\begin{aligned} \text{var}(a_1 v_1 + \cdots + a_n v_n \bowtie b) &= \{v_1, \ldots, v_n\} \\ \text{var}(\neg\phi) &= \text{var}(\phi) \\ \text{var}(\phi \wedge \phi) &= \text{var}(\phi) \cup \text{var}(\phi) \\ \text{var}(\phi \vee \phi) &= \text{var}(\phi) \cup \text{var}(\phi) \end{aligned} \tag{5.1}$$

In other words, a critical formula is a boolean combination of linear inequalities of variables observed from the state the monitored CPS. We use a standard semantic interpretation, defined as follows.

Definition 5.4. Given a partial assignment c and a formula ϕ such that $\mathrm{var}(\phi) \subseteq \mathrm{dom}(c)$, the assignment c *satisfies* (or *is a model of*) the formula ϕ, denoted by $c \models \phi$, when inductively:

$$\begin{aligned}
c &\models \sum_i a_i v_i \bowtie b && \text{iff} \quad \sum_i a_i c(v_i) \bowtie b \\
c &\models \neg\phi && \text{iff} \quad c \not\models \phi \\
c &\models \phi_1 \wedge \phi_2 && \text{iff} \quad c \models \phi_1 \text{ and } c \models \phi_2 \\
c &\models \phi_1 \vee \phi_2 && \text{iff} \quad c \models \phi_1 \text{ or } c \models \phi_2
\end{aligned} \tag{5.2}$$

The set of states satisfying a formula ϕ is denoted by $\mathcal{S}(\phi)$.

Proposition 5.1. *Let s be a state assignment and ϕ a critical condition formula. Determining if the state is critical, i.e. whether $s \models \phi$, can be implemented from Definition 5.4.*

Proof. The condition $\mathrm{var}(\phi) \subseteq \mathrm{dom}(s)$ holds because $\mathrm{dom}(s) = \mathcal{V}$ by the definition of state. This implies that $s \models \phi$ is well defined according to Definition 5.4. The condition $\mathrm{var}(s) \subseteq \mathrm{dom}(s)$ allows to verify the atomic expression $\sum_i a_i s(v_i) \bowtie b$. The other syntax cases can be evaluated by recursion since the $\models$ relation is defined inductively on the syntax of ϕ, which implies termination.

□

This implies that in case of full observability the implementation of the detection of critical conditions derives straightforwardly from Definition 5.4. Next section presents the proximity-based notion of criticality and possible computational methods.

Example 5.1. *The boiler example described in Section 4.2 uses the following variables:*

- *T_1, T_2, S_1, S_2, O: real variables for internal temperatures, desired temperatures (setpoints), and outdoor temperature.*

  Range constraints are defined from the domain, e.g. $[-10, 50]$ °C for ambient temperatures.

  The observation method is deep packet inspection of the Modbus protocol, selecting read response packets using packet headers and extracting sensor values from the packet payloads.

- *H: boolean variable corresponding to the on/off status of the heater. The range constraint is clearly $\{0, 1\}$ and the observation method is the same as the temperature variables.*

Notice that the critical conditions that refers to the unobservable variable M are not suitable for the framework presented in this chapter.

5.4 PREDICTIVE DETECTION OF CRITICAL CONDITIONS

The quantitative notion of the criticality of the current state of the system with respect to a critical condition specification is hereafter defined. It is based on a notion of proximity which uses standard definitions of metric spaces summarised in Section 1.2.

Proposition 5.2. *Let $n = \#\mathcal{V}$ be the number of the state variables. Any metric d on $\mathbb{R}^n$ induces a metric on the set of states $\mathcal{S}$ and a premetric on the parts of the states $2^{\mathcal{S}}$.*

Proof. CPS variables defined in Section 5.2 range on real numbers. Let v be any enumeration of the variables, i.e. a bijective function $v\colon \{1,\dots,n\} \to \mathcal{V}$. Let

$$R = \{x \in \mathbb{R}^n \mid \forall i.\, x_i \in \mathrm{range}(v(i))\}$$

be the subset of $\mathbb{R}^n$ consistent with the range of the variables. Notice that (R, d) is a metric space. We prove R and $\mathcal{S}$ are isomorphic by defining a bijection. The injective function $r\colon \mathcal{S} \to R$ defined as $r(s) = \big(s(v(1)),\dots,s(v(n))\big)$ is bijective: for each $x \in R$ there exists a unique CPS state s defined as $s(v(i)) = x_i$ for $i = 1 \dots n$. Since (R, d) is a metric space, $(\mathcal{S}, \hat{d})$ is also a metric space where $\hat{d}(s,t) = d(r(s), r(t))$. The metric $\hat{d}$ induces a premetric $\hat{D}$ on $2^{\mathcal{S}}$ as in Proposition 1.1. With a little abuse of notation, in the following we will use d and D to denote also $\hat{d}$ and $\hat{D}$. □

Given a metric d on $\mathbb{R}^n$, where $n = \#\mathcal{V}$, we use the induced premetric D to define the quantitative notion of criticality of a state s with respect to a critical condition ϕ as follows.

Definition 5.5. Let D be a premetric on $\mathcal{S}$. The *criticality* of a CPS state s from the critical condition ϕ is defined as

$$C(s,\phi) := \begin{cases} D(s,\neg\phi) & \text{if } s \models \phi \\ -D(s,\phi) & \text{if } s \not\models \phi \end{cases} \tag{5.3}$$

The criticality value is positive when the state s is critical and indicates how far the state is from the licit states. Otherwise, it is negative and indicates how close the state is to critical states.

Definition 5.5 is parametric with respect to the chosen metric d on the set of states $\mathcal{S}$ associated to ϕ. Table 5.1 contains possible examples of metrics and their combinations. The chosen metric is part of the critical condition specification. The actual choice depends on the application, as the following example shows.

$m_V(s,t) = \sum_{v \in V} \lvert s(v) - t(v) \rvert$	Manhattan distance (i.e. L_1 metric on $\mathbb{R}^n$)
$wm_V(s,t) = \sum_{v \in V} w_v \lvert s(v) - t(v) \rvert$	Weighted Manhattan distance ($w_v \geq 0$)
$nm_V(s,t) = \frac{1}{\#V} \sum_{v \in V} \frac{\lvert s(v) - t(v) \rvert}{v_{\max} - v_{\min}}$	Normalised Manhattan distance (defined if $v_{\min}, v_{\min} \in \mathbb{R}$)
$h_V(s,t) = \#\{v \in V \mid s(v) \neq t(v)\}$	Hamming distance
$wh_V(s,t) = \sum_{\substack{v \in V \\ s(v) \neq t(v)}} w_v$	Weighted Hamming distance ($w_v \geq 0$)
$nh_V(s,t) = \frac{1}{\#V} h_V(s,t)$	Normalised Hamming distance
$\underset{d_V, d_U}{\mathrm{comb}}(s,t) = d_V(s_{\mid V}, t_{\mid V}) + d_U(s_{\mid U}, t_{\mid U})$	Combination of metrics d_V and d_U

where $s, t \in \mathcal{S}$, $V, U \subseteq \mathcal{V}$, $v_{\min} = \min(\mathrm{range}(v))$, $v_{\min} = \max(\mathrm{range}(v))$, and the notation $s_{\mid X}$ denotes the standard projection of the vector s on X.

Table 5.1: Example of metrics on $\mathcal{S}$.

Example 5.2. *Equation* (4.1) *in the chemical process example of Section 4.3 is defined as*

$$\phi = RF_A \neq rf_A \vee RF_B \neq rf_B \vee RF_C \neq rf_C \vee RF_L \neq rf_L \vee RF_R \neq rf_R$$

and is satisfied by observations not compliant with the expected number of read messages per time unit captured from the network traffic.

The Manhattan metric m_V defined in Table 5.1, with $V = \mathrm{var}(\phi)$, induces a premetric D_m on the states $\mathcal{S}$ that captures the actual differences of the observed values from the expected ones. In particular, given a state s defined as $s(RF_A) = rf_A + 10$, $s(RF_B) = rf_B - 15$, and $s(RF_X) = rf_X$ for $X \in \{C, L, R\}$, the criticality $C_m(s, \phi) = 25$.

The Hamming metric h_V instead induces a premetric D_h that captures the number of variables not compliant with the specification. Indeed, given the same state s, the criticality $C_h(s, \phi) = 2$.

Different metrics can also be combined on different variables. The operator comb_{d_V, d_U} *in Table 5.1 shows a possible combination of d_V and d_U.*

Figure 5.2 shows the flow of the reasoner. At each iteration, it receives the state s from the observer, it checks if $s \models \phi$ using Definition 5.4 and Proposition 5.1, then it compute either $D(s, \neg\phi)$ or $-D(s, \phi)$.

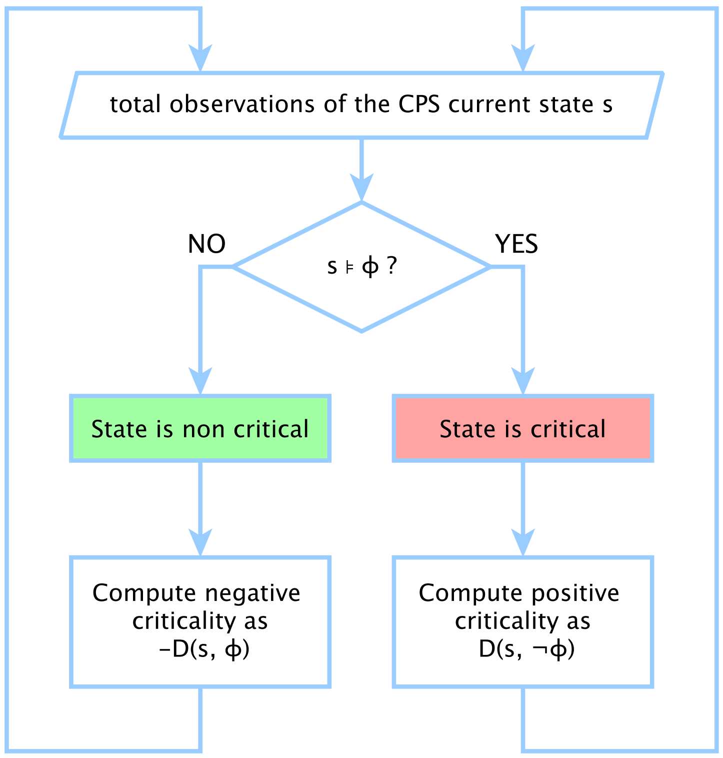

total observations of the CPS current state s
s ⊧ ϕ ?
NO
YES
State is non critical
State is critical
Compute negative criticality as −D(s, ϕ)
Compute positive criticality as D(s, ¬ϕ)

Figure 5.2: Evaluation of the criticality of current state.

The following sections present different techniques to evaluate $D(s, \phi)$ for any given ϕ. The threshold-based approach, presented in Section 5.5, does not need to actually compute the quantity $D(s, \phi)$, but only verifies if $D(s, \phi) < l_i$ for a given set of thresholds l_i. Section 5.6 presents some techniques to compute the distance, based on linear programming and on SMT. The first approach is computationally easier but inherently approximated, while the second approach needs to solve an optimisation problem.

5.5 BEST-THRESHOLD APPROXIMATION OF CRITICALITY

This section presents the method we developed to evaluate an approximation $D(s, \phi)$ without computing it. It requires that any critical specification includes a set of real thresholds $\{l_1, \dots, l_h\}$, where $l_1 < l_2 < \dots < l_h$. The set represents different levels of alerting defined by an operator. To ease the presentation, we define $l_0 = -\infty$ and $l_{h+1} = +\infty$.

The approach searches for the best threshold approximation, i.e. the $i \in \{0, \dots, h\}$ such that

$$l_i \leq D(s, \phi) < l_{i+1} \tag{5.4}$$

The technique is inspired by the geometrical and topological properties of $\mathcal{S}(\phi)$. We prove some required linear algebra results about convex polyhedra, present the algorithm for finding the threshold approximation based on polyhedra manipulation, and prove the correctness of the algorithm. The algorithm uses some of the computational geometry operations provided by Parma Polyhedra Library, described in Section 1.2. While the definition of $D(s,\phi)$ is generally parametric in the chosen metric, the threshold-based approach presented in this section is restricted to the Manhattan distance and its induced premetric.

Proposition 5.3. *Given a critical formula ϕ, the set $\mathcal{S}(\phi)$ is the union of a finite number of convex polyhedra.*

Proof. For any formula ϕ, it is a well know result that there exists an equivalent ϕ' which is in Disjunctive Normal Form (DNF), i.e.

$$\phi' = \bigvee_i \bigwedge_j H_{ij} \tag{5.5}$$

where H_{ij} are atomic formulae. The DNF formula is obtained by repeatedly applying De Morgan laws and double negations simplification. From the grammar in Definition 5.3, i.e. $H_{ij} = \sum_k a_k v_k \leq b$, $a_k, b \in \mathbb{R}$, and $v_k \in \mathcal{V}$. Thus, $\mathcal{S}(H_{ij})$ are affine half-spaces (or linear inequalities) as defined in Section 1.2. This implies that $P_i = \mathcal{S}(\bigwedge_j H_{ij}) = \bigcap_j \mathcal{S}(H_{ij})$ are convex polyhedra and $\mathcal{S}(\bigvee_i \bigwedge_j H_{ij}) = \bigcup_i P_i$. □

Proposition 5.4. *Let $x \in \mathbb{R}^n$ be a real vector and $l \in \mathbb{R}^+$ a non negative real value. Let $u_i = x + le^{(i)}$ and $v_i = x - le^{(i)}$, where vectors $e^{(i)}$ are the standard base[1] of the vectorial space $\mathbb{R}^n$.*

For any $y \in \mathbb{R}^n$

$$|x - y| \leq l \quad \textit{iff} \quad y \in \text{convexhull}(\{u_i\} \cup \{v_i\}) \tag{5.6}$$

where convexhull *is defined as in Section 1.2 and $|\cdot|$ is the l_1-norm on $\mathbb{R}^n$.*

Proof. We prove the result algebraically, while Figure 5.3 depicts the geometrical $\mathbb{R}^2$ case.

1 $e^{(i)}$ is the vector of $\mathbb{R}^n$ defined as $(e^{(i)})_i = 1$ and $(e^{(i)})_j = 0$ for all $i \neq j$.

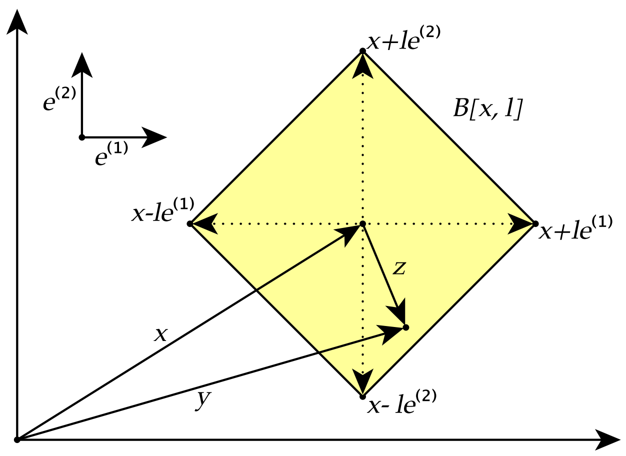


Figure 5.3: Metric ball $B[x,l]$ as a convex hull of $x \pm le^{(i)}$.

($\Rightarrow$) Let y be any vector such that $|x - y| \leq l$. We show that y is a convex combination of u_i and v_i, i.e. that there exist real parameters α_i and β_i such that

$$\alpha_i, \beta_i \geq 0 \tag{5.7}$$

$$\sum_{i=1}^{n} \alpha_i + \sum_{i=1}^{n} \beta_i = 1 \tag{5.8}$$

$$y = \sum_{i=1}^{n} \alpha_i (x + le^{(i)}) + \sum_{i=1}^{n} \beta_i (x - le^{(i)}) \tag{5.9}$$

Let $z = y - x$. By hypothesis $|z| = \sum_i |z_i| \leq l$.
Define α_i and β_i as

$$\alpha_i = \frac{1}{2}\left(\frac{|z_i|}{|z|} + \frac{z_i}{l}\right) \qquad \beta_i = \frac{1}{2}\left(\frac{|z_i|}{|z|} - \frac{z_i}{l}\right) \tag{5.10}$$

Equation (5.7) holds: if $z_i \geq 0$ clearly $\alpha_i \geq 0$ and

$$\beta_i = \frac{|z_i|}{|z|} - \frac{z_i}{l} = \frac{|z_i|}{|z|} - \frac{|z_i|}{l} \geq \frac{|z_i|}{l} - \frac{|z_i|}{l} = 0$$

as $|z| \leq l$. The case $z_i \leq 0$ is symmetrical.
Equation (5.8) holds by construction since

$$\begin{aligned}
\sum_{i=1}^{n} \alpha_i + \sum_{i=1}^{n} \beta_i &= \sum_{i=1}^{n} \frac{1}{2}\left(\frac{|z_i|}{|z|} + \frac{z_i}{l}\right) + \sum_{i=1}^{n} \frac{1}{2}\left(\frac{|z_i|}{|z|} - \frac{z_i}{l}\right) \\
&= \sum_{i=1}^{n} \frac{1}{2}\left(\frac{|z_i|}{|z|} + \frac{z_i}{l}\right) + \sum_{i=1}^{n} \frac{1}{2}\left(\frac{|z_i|}{|z|} - \frac{z_i}{l}\right) \\
&= \sum_{i=1}^{n} \frac{|z_i|}{|z|} = 1
\end{aligned}$$

The following calculation proves Eq. (5.9)

$$\begin{aligned}
&\sum_{i=1}^{n} \alpha_i(x + le^{(i)}) + \sum_{i=1}^{n} \beta_i(x - le^{(i)}) \\
&= \sum_{i=1}^{n} \alpha_i x + \sum_{i=1}^{n} \beta_i x + \sum_{i=1}^{n} \alpha_i le^{(i)} - \sum_{i=1}^{n} \beta_i le^{(i)} \\
&= x + \sum_{i=1}^{n} \frac{1}{2}\left(\frac{|z_i|}{|z|} + \frac{z_i}{l}\right) le^{(i)} - \sum_{i=1}^{n} \frac{1}{2}\left(\frac{|z_i|}{|z|} - \frac{z_i}{l}\right) le^{(i)} \\
&= x + \frac{1}{2}\sum_{i=1}^{n} \frac{|z_i|}{|z|} + \frac{1}{2}\sum_{i=1}^{n} z_i e^{(i)} - \frac{1}{2}\sum_{i=1}^{n} \frac{|z_i|}{|z|} + \frac{1}{2}\sum_{i=1}^{n} z_i e^{(i)} \\
&= x + \sum_{i=1}^{n} z_i e^{(i)} = x + z = y
\end{aligned}$$

($\Leftarrow$) Let y be a vector that belongs to convexhull$(\{u_i\} \cup \{v_i\})$, Thus, there exists $\alpha_i, \beta_i \geq 0$ such that $\sum_i \alpha_i + \sum_i \beta_i = 1$ and

$$y = \sum_{i=1}^{n} \alpha_i(x + le^{(i)}) + \sum_{i=1}^{n} \beta_i(x - le^{(i)})$$

Then

$$\begin{aligned}
|y - x| &= \left|\sum_{i=1}^{n} \alpha_i(x + le^{(i)}) + \sum_{i=1}^{n} \beta_i(x - le^{(i)}) - x\right| \\
&= \left|\sum_{i=1}^{n} \alpha_i x + \sum_{i=1}^{n} \beta_i x + \sum_{i=1}^{n} \alpha_i le^{(i)} - \sum_{i=1}^{n} \beta_i le^{(i)} - x\right| \\
&= \left|l\sum_{i=1}^{n} (\alpha_i - \beta_i)e^{(i)}\right| = l\sum_{i=1}^{n} |\alpha_i - \beta_i| \leq l\sum_{i=1}^{n} |\alpha_i| + |\beta_i| = l
\end{aligned}$$

since $\sum_i w_i e^{(i)} = w$ for all vectors $w \in \mathbb{R}^n$ by definition of $e^{(i)}$. □

Proposition 5.5. *Given a state s, a critical formula ϕ, and a real value l,*

$$D(s, \phi) \leq l \quad \textit{iff} \quad \text{convexhull}(\{s \pm le^{(i)}\}) \cap \mathcal{S}(\phi) \neq \emptyset \qquad (5.11)$$

where D is the premetric induced by the Manhattan metric d.

Proof. $D(s, \phi) \leq l$ iff there exists a state $t \in \mathcal{S}(\phi)$ such that $d(s, t) \leq l$ by definition of D. By Proposition 5.4, this is equivalent to $t \in$ convexhull$(\{s \pm le^{(i)}\})$ which is equivalent to the thesis. □

We developed Algorithm 5.1 guided by the previous geometrical properties of polyhedra using the PPL library [1]. The Init function takes a critical formula ϕ in DNF form and build the set of polyhedra P_m, such that $\mathcal{S}(\phi) = \bigcup_m P_m$. It uses the PPL method *from_constraints*, described in Section 1.2, which takes the linear inequalities occurring

Algorithm 5.1 PPL-based Distance Threshold.

```
Require: (same as Algorithm 5.1)
  State $s$ as a vector of $\mathbb{R}^n$
  DNF formula $\phi = \bigvee_m \psi_m$
    conjunctive clauses $\psi_m = \bigwedge_j H_j$
    atomic linear constraint $H_j = \sum_{v_i \in \mathcal{V}} a_{ji} v_i \leq b_j$
  metric used: Manhattan
  function INIT($\phi$)
    for $\psi_m \in \phi$ do
      $P_m \leftarrow$ ppl.from_constraints($\psi_m$)        ▷ polyhedra $P_m$ are global
  function TRESHOLDDISTANCE($s$, $l$)
    $S \leftarrow \{s \pm le^{(j)} \mid j = 1 \dots n\}$        ▷ where $n = \#\mathcal{V}$
    sball $\leftarrow$ ppl.from_generators($S, \emptyset, \emptyset$)
    for each $P_m$ do
      if sball.intersection($P_m$).isempty() then
        return True
    return False
  function SEARCH($s$, $\{\{l_1, \dots, l_h\}$)
    for $i = i \dots h$ do
      if THRESHOLDDISTANCE($s$, $l_i$) then
        return $l_i$
    return $+\infty$
```

where $e^{(j)}$ are the unit vectors of the standard basis of $\mathbb{R}^n$. The pseudo-code is based on the Parma Polyhedra Library described in Section 1.2.

in ψ_m and generates the corresponding representation of the polyhedron. Notice that the DNF formula exists for every ϕ but the conversion in DNF has an exponential time complexity with respect to the number of atomic formula in the worst cases. This is a well known result. However, the conversion is done only once at initialisation time and has no impact on the performances of the monitor at runtime after the initialisation.

Once the polyhedra representation $\{P_m\}$ is created from ϕ, at runtime the function THRESHOLDDISTANCE determines if $D(s, P_m) \leq l$ for some P_m. This is achieved constructing the convex hull S of vectors $\{s \pm le^{(i)}\}$ using the PPL method *from_generators* described in Section 1.2, where the sets of rays R and lines L passed as input arguments are empty. The function THRESHOLDDISTANCE(s, l) returns *true* if the convex hull S intersects some P_m. This is correct according to Eq. (5.11). The search for the best threshold approximation is achieved through a linear search of the smallest threshold l_i such that $D(s, P_m) \leq l_i$ for some P_m. The correctness of the algorithm comes directly from

Proposition 5.4, from Eq. (5.11), and from the correctness of the PPL library.

5.6 COMPUTATION OF CRITICALITY

This section shows possible methods to compute $D(s, \phi)$ defined in previous sections and used to compute the criticality $C(s, \phi)$ of the current observed state s of the CPS with respect to a given critical condition ϕ.

Linear Programming-based Approach

The method described in this section uses linear programming and is restricted to use the weighted Manhattan metric of Table 5.1 and its induced premetric simply denoted by D.

This approach requires a DNF critical formula as input. Also in this case, a general formula is converted to an equivalent DNF only during the initialisation phase, and the possible high computational cost of the operation has no impact at runtime.

We assume the same notation for the DNF formula $\phi = \bigvee_m \psi_m$, $\psi_m = \bigwedge_j H_j$, and $H_j = \sum_i a_i v_i \leq b$, where $i = 1 \ldots n$ and $n = \#\mathcal{V}$ is the number of variables.

Proposition 5.6. *Given a state s and a DNF formula $\phi = \bigvee_m \psi_m$, then*

$$D(s, \phi) = \min_m D(s, \psi_m) \tag{5.12}$$

Proof.

$$\begin{aligned} D(s, \phi) &= D(s, \mathcal{S}(\bigvee_m \psi_m)) = D(s, \bigcup_m \mathcal{S}(\psi_m)) \\ &= \min_m D(s, \mathcal{S}(\psi_m)) = \min_m D(s, \psi_m) \end{aligned}$$

□

Thus, to compute $D(s, \phi)$ it is sufficient to develop a method for computing $D(s, \psi_m)$, where ψ_m is a conjunction of linear constraints, and to iterate on m to find the minimum value $D(s, \psi_m)$.

Using the weighted Manhattan distance on a subset of variables $V \subseteq \mathcal{V}$,we have

$$D(s, \psi_m) = \min_{t \in \mathcal{S}(\psi_m)} \sum_{v_i \in V} w_i |s(v_i) - t(v_i)|$$

Hence, computing $D(s, \psi_m)$ is an optimisation problem on linear constraints. It is known that this kind of optimisation objective can be converted to a linear expression eliminating the absolute value and adding additional variables. Precisely, for any real number x, its absolute value $|x|$ is the minimum real u such that $-u \leq x \leq u$ (this implies $u \geq 0$).

The equivalent linear optimisation problem is defined using two vectors $t \in \mathbb{R}^n$ and $u \in \mathbb{R}^n$ where $n = \#V$ is the number of variables:

$$\text{goal:} \quad \text{minimise} \sum_{i=1}^{h} w_i u_i$$

$$\text{constraints:} \quad \begin{cases} \sum_i a_{ji} t_i \leq b_j & \text{for all } i \text{ s.t. } H_j \text{ in } \psi_m \\ s_i - t_i \leq u_i & \text{for all } i \text{ s.t. } v_i \in V \\ -u_i \leq s_i - t_i & \text{for all } i \text{ s.t. } v_i \in V \end{cases}$$

where s_i is a constant of the problem for any given state s, a_{ji} and b_j are constants expressed in the formula ψ_m, while t_i and u_i are the variables of the problem.

The goal $\min \sum_{i=1}^{h} w_i u_i = D(s, \psi_m)$ can be computed with any linear programming solver. Proposition 5.6 implies the correctness of the approach.

Satisfiability Modulo Theory-based Approach

This sections describes different SMT-based methods to compute the distance $D(s, \phi)$.

SMT WITH UNIVERSAL QUANTIFICATION. This section proposes a method to compute the distance $D(s, \phi)$ directly from the logical description of the computation problem. Let s be the current CPS state, i.e. the total assignment of the observed values to the variables as defined in Definition 5.1, and ϕ a state formula. The proximity $D(s, \phi)$ is defined as the minimal real value $d(s, t)$ such that $t \models \phi$ for some state t. The minimality of the distance can be expressed as the problem of determining whether there exists a state t such that

$$t \models \phi \quad \wedge \quad \forall x \in \mathcal{S}.\, x \models \phi \rightarrow d(s,t) \leq d(s,x) \qquad (5.13)$$

A universal quantification over the set of states occurs in Eq. (5.13). Thus, any solver that include the Linear Integer and Rational Arithmetic (LIRA) with universal quantification accepts Eq. (5.13) as input and returns the state t that satisfies the assertion. Given such t, it is possible to compute $D(s, \phi) = d(s, t)$. Algorithm 5.2 shows the pseudo-code of this method, where *metric* is the arithmetic expression that denotes the chosen metric on $\mathcal{S}$.

Algorithm 5.2 Quantifier SMT computation of the distance.

```
function DISTANCE($s$, $\phi$)
    T ← a set of fresh symbols for variables
    solver ← new SMT-Solver
    solver.assert $\phi[\mathcal{V} \mapsto \mathrm{T}]$
    solver.assert $\forall X.\, \phi[\mathcal{V} \mapsto \mathrm{X}] \rightarrow \mathrm{metric(s, T)} \leq \mathrm{metric(s, X)}$
    model ← solver.check-sat()
    if model not found then
        return Error: $\phi$ is unsatisfiable
    else
        return model.getvalue(metric(s, T))
```

A set T of fresh variable symbols is used to represent the state t to be found. The algorithm creates a new instance of the SMT solver, and asserts the logical formula $\phi[\mathcal{V} \mapsto T]$, that is the formula ϕ where each variable is replaced with the corresponding symbol in T. Notice that this assertion is satisfiable if and only if ϕ is satisfiable. Then, it adds the assertion $\forall X.\, \phi[\mathcal{V} \mapsto \mathrm{X}] \rightarrow \mathrm{metric(s, T)} \leq \mathrm{metric(s, X)}$.

Notice that the expression $\mathrm{metric(s, T)} \leq \mathrm{metric(s, X)}$ is always satisfiable because it is true when T and X are assigned to the same values. Thus, $\phi[\mathcal{V} \mapsto \mathrm{X}] \rightarrow \mathrm{metric(s, T)} \leq \mathrm{metric(s, X)}$ is always satisfiable as a consequence.

As a consequence, the two assertions are satisfiable if and only if ϕ is satisfiable. In this case, the solver finds a model, i.e. an assignment t of values to each symbol in T, and returns the value of $d(s, t)$ as a result in the last line of the algorithm. Notice that an unsatisfiable critical condition ϕ can be considered a mistake of the human operator that defined the formula.

Quantifier-free Iterative SMT

This sections describes an SMT-based method to compute $D(s, \phi)$ which uses the quantifier-free linear integer and rational arithmetic (QF_LIRA). While quantifier-free arithmetic is less expressive, SMT solvers for such fragment are much more efficient in terms of computational complexity and performance. For this reason, avoiding the use of quantifiers is usually a good practice. However, this also needs to overcome the lower expressibility of the logic with tailored algorithms.

Algorithm 5.3 shows the pseudo-code based on an iterative search on the distance to minimise. Intuitively, in order to compute

$$D(s, \phi) = \inf_{t \models \phi} d(s, t)$$

at each iteration the algorithm search for any critical state t, i.e. a state t which is a model of ϕ. If found, it computes the value $d(s, t)$ of the current approximation of the distance, and add an assertion $d < d(s, t) - \epsilon$ for a given error tolerance ϵ. If the solver finds a model in the next iteration, then the new critical state t is necessarily closer to s. The algorithm ends when no model is found, and the last model found represent the critical state that minimise the distance from s. Termination is guaranteed by the presence of a small positive value ϵ in the assertion $d < d(s, t) - \epsilon$. It is worth noticing that the presence of ϵ introduce an approximation error. Indeed, it may be the case that the last found approximation of the distance is higher than the actual distance. However, it can be proved that inequality

$$|D(s, \phi) - d| < \epsilon \qquad (5.14)$$

always holds, where d is the final result of the algorithm. Thus, the value ϵ represents the maximum accepted error in the final result.

The algorithm uses a set of fresh variables T to represent the state t. The term metric(s, T) represents the arithmetic expression of the distance between the state s, which is a given vector of values and a constant of the problem, and the state t represented by T. SMT solvers provide arithmetic operator, boolean operators, and the conditional operator that are sufficient to express all the metrics in Table 5.1. If the solver fails at the first check for satisfiability, than the critical formula ϕ is not satisfiable, i.e. the security operator made a mistake in the definition of ϕ because the detection of an unsatisfiable critical condition is meaningless.

Algorithm 5.3 SMT iterative computation of the distance.

```
function DISTANCE($s$, $\phi$)
    T ← a set of fresh symbols for variables
    d ← new real symbol
    solver ← new SMT-Solver
    solver.assert $\phi[\mathcal{V} \mapsto \mathrm{T}]$
    solver.assert d = metric(s, T)
    model ← solver.check-sat()
    if model not found then
        return Error: $\phi$ is unsatisfiable
    else
        while model found do
            distance ← model.getvalue(d)
            solver.assert d < distance - $\epsilon$
            model ← solver.check-sat()
    return model.getvalue(d)
```

6 THE CORE FRAMEWORK WITH UNOBSERVABLES AND OBSERVATION TIMES

6.1 OVERVIEW

A number of works in the literature about attack detection and monitoring for CPS assume that the aspects of interest of the monitored system are observable and develop techniques to extract and analyse these features, as described in Section 3.3. However, the assumption that it is always possible to retrieve the value of all the variables might be too strong in real cases. This chapter relaxes this assumption and presents the core framework we developed to handle the variables that are unobservable but still required to specify known critical conditions. This capability appears as a novelty from the literature review.

Section 3.3 shows that features from the temporal behaviour of the CPS brings added value when it comes to attack or anomaly detections. The literature shows a large number of works for runtime verification of temporal specifications [4, 5, 7–9, 11, 55]. Our framework is capable of express simple temporal critical conditions, based on the observation times of the variables. The application of verification of temporal properties to our framework is far from being complete.

The main differences from the core framework described in the previous chapter follow.

- The specification of variables remains the same, but we now allow the observation method to be omitted or to fail. This captures permanently or temporarily unobservable variables.
- As a consequence, the observer passes a partial assignment of variables to the reasoner instead of a system state.
- Critical formulae enable indicating the time an observation occurred, the current time, and linear constraints on those times.
- The semantics of critical formulae is modified accordingly.
- In Chapter 5, it is always possible to determine whether $s \models \phi$ and the implementation is straightforward from the definition. With unobservables, this is impossible in the general case due to the missing values. We developed a different core of the reasoner that leverages SMT solvers to overcome these situations. The techniques based on polyhedra manipulations and linear programming do not apply to partial observations.
- The reasoner now accepts optional additional information about the system from human operators, sssas a refinement of partial observations. The information is specified in the same logical language of critical formulae. This interaction with operators is inspired to practices employed in security operation centres, where experts constantly interacts with tools and SIEM solutions.
- When the reasoner cannot discriminate if the current state is in a critical condition, it computes a logical conditions about system variables that implies the state is not critical. The condition is handed to human operators and called *assisted check*. In a sense, the reasoner suggests the operator to check some conditions about unobservables to better evaluate the criticality measure of the current state of the system. To limit the human effort, tt is important that this logical condition is kept as small as possible to avoid unnecessary conditions that the reasoner can solve with available information. Although we cannot state that our technique computes the minimal assisted check in a formal sense, we use simplification tactics and logical interpolants, which are meant to discard part of the unnecessary atoms in the assisted check.
- The proximity-based notion of criticality is identical, but it is not possible to compute it in the general case of unobservable

variables. The reasoner now computes a range of criticality values which represent the best and worst cases.

6.2 critical condition specification

Unobservable variables complicate the framework but allow for a greater expressiveness and practical feasibility. There are three main cases in which a variable is considered unobservable:

1. a variable bound to the value of a malfunctioning sensor that cannot provide its value;
2. a variable bound to a parameter of the CPS which is required to express the critical condition but that can never be observed by design, e.g. the temperature of a gas in a point where no thermometer has been installed;
3. any aspect of the monitored system that is inherently unobservable, e.g. the intention of a human operator that acts without specifying his actions in advance.

The specification of variables presents little differences with respect to Section 5.3. Each variable is associated to a name, a type, a range constraints, and an observation method as before, but the observation method is optional to reflect variables that are permanently unobservable. Moreover, the observation method is allowed to fail, to capture the case of variables that are temporarily unobservables. Precisely, a variable is considered unobservable either when the observation method is not provided or when it fails at run-time during observations. Another important difference is that each observed value is associated to a timestamp which represent the time at which the observation occurred.

Figure 6.1 depicts the main architecture of the framework modified for handling unobservable variables. Iteratively, the reasoner receives the observation o, a critical condition ϕ, and checks o against ϕ. If the critical condition only contains observable variables, the reasoner is always able to tell whether the CPS has reached the criticality or not. In presence of unobservable variables, it might be impossible to discriminate whether the CPS is in a critical state only from observations.

The reasoner is also able to take as input some further information about the CPS state in form of a logical assertion, hereafter called *refinement* and denoted by ρ. Refinements are typically provided by

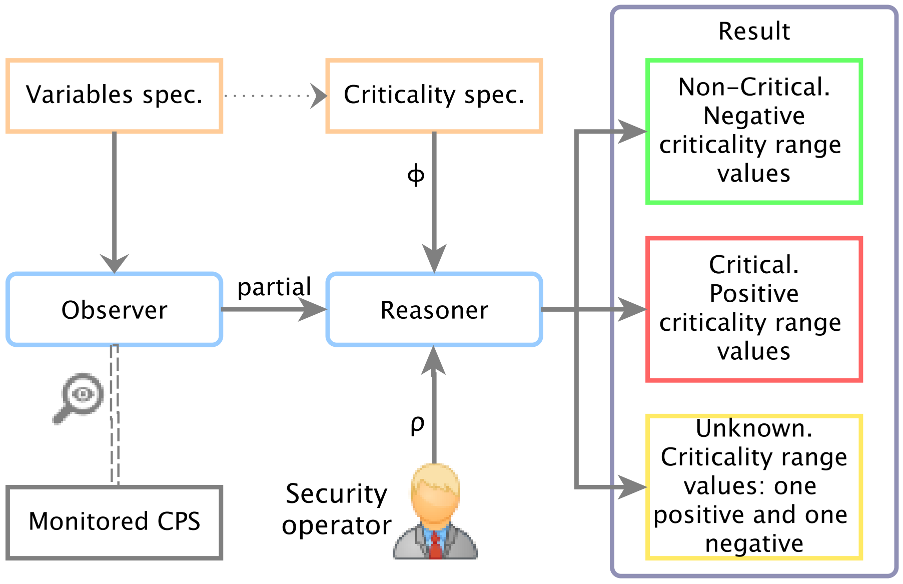

Variables spec.
Criticality spec.
ϕ
Observer
partial
Reasoner
ρ
Monitored CPS
Security operator
Result
Non-Critical. Negative criticality range values
Critical. Positive criticality range values
Unknown. Criticality range values: one positive and one negative

Figure 6.1: Structure of the real-time monitoring framework.

human operators to give the monitor additional information about unobservable variables.

When the reasoner is unable to determine whether the current state satisfies a critical condition, it computes a logical condition γ of unobservable variables that is sufficient to determine that the system state is not critical. The condition γ is hereafter called *assisted check*, because it is computed by the framework to help security operators figure out the missing unobservable information. In other words, the assisted check can guide operators to provide better knowledge refinements.

We use the same notion of variables $\mathcal{V}$ with the optional or possibly failing observation method. An observable variable is associated to timestamp values. In the following $\mathbb{T}$ denotes the real domain of time for notation purpose, i.e. $\mathbb{T} = \mathbb{R}$.

Definition 6.1. An *observation* is a partial mapping from variables to timestamped values. Formally, let $V \subseteq \mathcal{V}$ be a subset of variables. An observation is a pair of functions $o\colon V \to \mathbb{R}$ and $o^t\colon V \to \mathbb{T}$ such that $o(v) \in \text{range}(v)$ for each variable $v \in V$. The notation $\text{dom}(o)$ denotes its domain V.

When clear from the context we use o to indicate the pair (o, o^t). A state is a total observation where each value has a timestamp.

Definition 6.2. A *state* s of the monitored CPS is a total observation function that maps all variables to timestamped values, i.e. an observation such that $\text{dom}(s) = \mathcal{V}$. Given an observation o, we define

$$\mathcal{S}(o) = \{s \in \mathcal{S} \mid \forall v \in \text{dom}(o)\colon s(v) = o(v) \wedge s^t(v) = o^t(v)\}$$

as the set of states that coincide with s.

The reasoner regularly receives the most recent observation o from the observer. If $v \notin \mathrm{dom}(o)$ variable v is unobservable, otherwise the value $o(v)$ was observed at time $o^t(v)$. This allows reasoning about the actual time the value was observed, crucial to express time relationships about observations of different variables.

Definition 6.3. A *critical condition formula* is defined by the grammar:

$$X ::= v \mid v.t \mid \mathrm{now} \tag{6.1}$$

$$\phi ::= a_1 X_1 + \cdots + a_n X_n \bowtie b \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi \tag{6.2}$$

where $v \in \mathcal{V}$, now $\notin \mathcal{V}$ is a distinct symbol from variables, $a_i, b \in \mathbb{R}$, $\bowtie \in \{=, \neq, <, \leq, >, \geq\}$. Given a variable v, $v.t$ represents the timestamp of its value. The set of variables occurring in a formula ϕ is denoted by $\mathrm{var}(\phi)$.

A critical condition formula is a boolean combination of linear inequalities of values and timestamps of observation of variables. It expresses a property of the most recent observation of the CPS state, where both observable and unobservable variables may occur in a formula. We use the standard interpretation of formulae over assignments.

The critical condition Eq. (4.4) of the chemical process example described in Section 4.3 uses the syntax of Definition 6.3:

$$\neg(pLSon.t < pLSoff.t \rightarrow 30m < pLSoff.t - pLSon.t < 31)$$

where $pLSon, pLSoff \in \mathcal{V}$ are variables representing the observation of on and off commands to pump LS.

Definition 6.4. Given an observation o, a point in time τ, and a formula ϕ such that $\mathrm{var}(\phi) \subseteq \mathrm{dom}(o)$, the observation o *satisfies* at time τ the formula ϕ, denoted by $o, \tau \models \phi$, when recursively:

$$[\![v]\!]_{o,\tau} = o(v) \qquad [\![v.t]\!]_{o,\tau} = o^t(v) \qquad [\![\mathrm{now}]\!]_{o,\tau} = \tau$$

$$\begin{aligned}
o, \tau \models \sum_i a_i X_i \bowtie b \quad &\text{iff} \quad \sum_i a_i [\![X_i]\!]_{o,\tau} \bowtie b \\
o, \tau \models \neg\phi \quad &\text{iff} \quad o, \tau \not\models \phi \\
o, \tau \models \phi_1 \wedge \phi_2 \quad &\text{iff} \quad o, \tau \models \phi_1 \text{ and } o, \tau \models \phi_2 \\
o, \tau \models \phi_1 \vee \phi_2 \quad &\text{iff} \quad o, \tau \models \phi_1 \text{ or } o, \tau \models \phi_2
\end{aligned}$$

The set of states satisfying a formula ϕ is denoted by $\mathcal{S}(\phi)$.

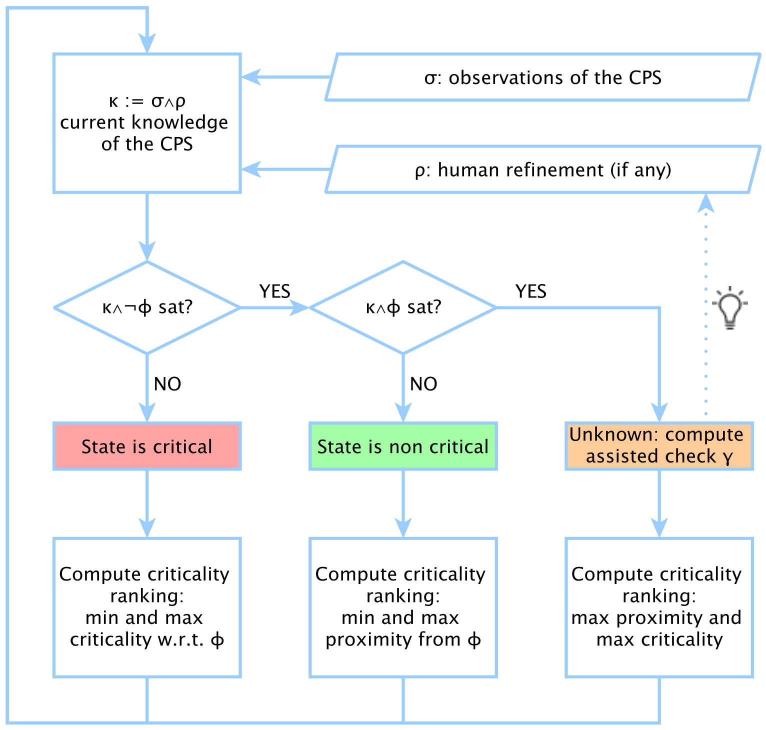

κ := σ∧ρ
current knowledge
of the CPS
σ: observations of the CPS
ρ: human refinement (if any)
κ∧¬ϕ sat?
YES
κ∧ϕ sat?
YES
NO
NO
State is critical
State is non critical
Unknown: compute assisted check γ
Compute criticality ranking: min and max criticality w.r.t. ϕ
Compute criticality ranking: min and max proximity from ϕ
Compute criticality ranking: max proximity and max criticality

Figure 6.2: Reasoner flow chart given criticality ϕ.

Our framework uses state formulae to define the known critical conditions of the monitored CPS. The reasoner iteratively receives from the observer the most recent observation o, and tries to evaluate if $o, \tau \models \phi$ for each critical condition ϕ where τ is the current time. In this way, the reasoning time can be different from the observation time, and the observation time for each variable can be different.

Notice that in the general case it is not possible to check if $o, \tau \models \phi$ due to unobservable variables. Formally, $o, \tau \models \phi$ can be simply checked using the semantics in Definition 6.4, defined by induction on the syntax of the formula, if and only if $\text{var}(\phi) \subseteq \text{dom}(o)$, i.e. an observed value is provided for each variables that occurs in ϕ. Otherwise, it may not be possible to check if ϕ is true in the current state. The following section describes how the reasoner handles these situations.

6.3 DETECTION OF CRITICAL CONDITIONS

Figure 6.2 depicts the behaviour of the reasoner. At each iteration it receives two inputs: an observation o from the observer and an optional information refinement ρ from the operator.

The core of the reasoner is an SMT engine. The reasoner computes a logical formula that is the equivalent of the observation as follows:

$$\sigma_o := \bigwedge_{v \in \mathrm{dom}(o)} v = o(v) \wedge v.t = o^t(v) \tag{6.3}$$

where v and $v.t$ are distinct symbols for the value and timestamp of variable v. Notice that unobservable variables, i.e. variables not defined in o, do not appear in σ_o. In the following we use σ to denote σ_o since the observation o is fixed for each iteration of the reasoner.

The refinement is a logical assertion that enables an operator to provide the reasoner with any further information about unobservable variables. It there is no such information then $\rho := \mathbf{true}$.

The logical expression $\kappa := \sigma \wedge \rho$ represents all the information that the reasoner knows about the current CPS state s. To discriminate if the CPS is currently in a critical state the reasoner checks whether the formulae $\kappa \wedge \phi$ and $\kappa \wedge \neg\phi$ are *satisfiable* using an SMT solver. Three cases are possible:

1. The system *is in a critical state*, regardless unobservable values, or equivalently $\mathcal{S}(\kappa) \cap \mathcal{S}(\phi)^{\complement} = \emptyset$. This is equivalent to checking whether the formula

$$\kappa \wedge \neg\phi \text{ is unsatisfiable.} \tag{6.4}$$

2. The system *is not in a critical state* regardless unobservable values, or equivalently $\mathcal{S}(\kappa) \cap \mathcal{S}(\phi) = \emptyset$. Similarly, this is equivalent to checking whether formula

$$\kappa \wedge \phi \text{ is unsatisfiable.} \tag{6.5}$$

3. If both formulae in (6.4) and (6.5) are satisfiable, then $\mathcal{S}(\kappa) \cap \mathcal{S}(\phi) \neq \emptyset$ and $\mathcal{S}(\kappa) \setminus \mathcal{S}(\phi) \neq \emptyset$. In other words, it is not possible to establish from κ whether the actual CPS state is critical, because this depends on some unobservable values not in κ.

In the first and second cases the reasoner can compute an estimation of the criticality of the current state respectively, as explained in Section 6.4.

In the third case κ does not contain enough information to discriminate whether the CPS is in critical state. A way to obtain a more precise result is to provide a more informative refinement than the one contained in κ.

In practical cases it can be hard for a human operator to understand which piece of information is missing. To this aim, the reasoner is able to calculate a condition γ that is sufficient to guarantee the non-criticality of the current CPS. Our framework can provide a human operator with γ as a guide for better refinements. Indeed, the operator can try verifying if γ holds, or at least if some of its subformulae. This way the operator may acquire some information, make educated assumptions on unobservable variables, and provide it back to the reasoner in the form of a more informative refinement. For this reason, the reasoner acts as an assistant to the human operator, and the formula γ is called *assisted check*. In practical cases, the operator must be able to handle the complexity of the assisted check formula, thus it is crucial that the size of γ is as small as possible.

A possible way to compute γ is hereafter described. Notice that the formula $\kappa \wedge \neg\phi$ is trivially an assisted check, but useless for finding better refinements. Moreover it contains all pairs of the form $v = x$ and $v.t = y$ for all observable variables v for some value x and y. It is useless for a human to check variables whose values has been retrieved by the observer.

We use the well known notion of Craig interpolant, provided by most SMT solvers, to compute the assisted check γ. Given two mutually unsatisfiable formulae α and β, a *Craig interpolant* (denoted by $\mathrm{interpolant}(\alpha, \beta)$) is a formula η such that $\mathrm{var}(\eta) \subseteq \mathrm{var}(\alpha) \cap \mathrm{var}(\beta)$ and formulae $\alpha \to \eta$ and $\eta \to \neg\beta$ are valid. In other words, the formula η is an explanation for the mutual unsatisfiability that uses only the variables that are common in α and β.

Definition 6.5 (Assisted check computation)**.** Let o be a partial observation, σ be defined as Eq. (6.3), ρ a refinement, and ϕ a critical formula such that both $\sigma \wedge \rho \wedge \neg\phi$ and $\sigma \wedge \rho \wedge \phi$ are satisfiable. Let ρ_o and ϕ_o be the formula obtained replacing each occurrence of v and $v.t$ with $o(v)$ and $o^t(v)$ resp. for all $v \in \mathrm{dom}(o)$. The *assisted check* is defined as

$$\gamma := \mathrm{interpolant}(\rho_o \wedge \neg\phi_o, \phi_o) \tag{6.6}$$

We now show that γ is an assisted check, i.e. implies the non criticality of the current state, and does not contain observable variables.

Proposition 6.1. *The formula γ defined in Eq.* (6.6) *is an assisted check that does not contain observable variables.*

Proof. Since $\gamma = \mathrm{interpolant}(\rho_o \wedge \neg\phi_o, \phi_o)$, by definition of Craig interpolant

$$\gamma \to \neg\phi_o \tag{6.7}$$

$$\mathrm{var}(\gamma) \subseteq \mathrm{var}(\rho_o \wedge \neg\phi_o) \cap \mathrm{var}(\phi_o) \tag{6.8}$$

Eq. (6.7) implies that $o, \tau \models \neg\phi_o$, which also implies that $o, \tau \models \neg\phi$ by Definition 6.4.

The set of observable variables $\mathrm{dom}(o) \cap \mathrm{var}(\phi_o) = \emptyset$ by construction of ϕ_o. Thus, Eq. (6.8) implies that $\mathrm{dom}(o) \cap \mathrm{var}\,\gamma = \emptyset$. □

Example 6.1. *Assume that the example CPS of Section 4.2 reaches a state where the room temperatures are* 15 °C *and* 23 °C*, the desired temperatures are* 21 °C *and* 17 °C*, the main controller sends a Modbus write message to the PLC controlling the water heater to turn it off, and the outdoor thermometer is temporarily broken (i.e. O is unobservable). The observer collects such information from the network traffic and provides the reasoner with*

$$\sigma := \neg H \wedge T_1 = 15 \wedge S_1 = 21 \wedge T_2 = 23 \wedge S_2 = 17 \tag{6.9}$$

Assume no further refinement is provided, i.e. $\kappa = \sigma$*, and consider the critical formula* ϕ_3 *defined in the example Section 4.2 as*

$$\phi_3 = \neg M \wedge \neg H \wedge \big((T_1 < S_1 \wedge O < S_1) \vee (T_2 \wedge O < S_2)\big)$$

In this example, our framework can verify that both formulae $\kappa \wedge \phi_3$ *and* $\kappa \wedge \neg\phi_3$ *are satisfiable. Indeed, the current knowledge* κ *does not contain enough information discriminate if the system state is critical, because this depends on the actual value of the unobservable variables M and O. In this case, the assisted check defined as Eq.* (6.6) *results*

$$\gamma := M \vee O \geq 21$$

The result is correct: to discriminate the criticality of the current state it is enough to check if the operator intentionally sent the command or if the outdoor temperature is greater than S_1*, which is 21. Notice that* γ *only contains the unobservable variables.*

6.4 PREDICTION OF CRITICAL CONDITIONS

In this section we use the same notion of proximity $D(s, \phi)$ described in the previous chapter.

When the current CPS state is critical, i.e. $\kappa \wedge \neg\phi$ is unsatisfiable, proximity $D(\kappa, \phi) = 0$. When the CPS is in a critical state, i.e. when $\kappa \wedge \phi$ is unsatisfiable, computing the proximity from the critical condition $D(\kappa, \phi)$ is an optimisation problem on linear constraints, since critical formulae κ and ϕ represent boolean combination of linear inequalities. Our framework uses SMT-based optimisation techniques, such as the one provided by the Z3 prover [34].

Due to unobservable variables, κ does not represent one system state but a set of possible states. It is possible to evaluate the proximity from ϕ or the criticality w.r.t. ϕ in the best and worst possible cases. The *criticality range* of κ with respect to ϕ is the pair $C(\kappa, \phi) = (C_{\min}, C_{\max})$ defined as

$$C_{\min}(\kappa, \phi) := \begin{cases} -D_{\max}(\kappa \wedge \neg\phi, \phi) & \text{if } \kappa \wedge \neg\phi \text{ is satisfiable} \\ D_{\min}(\kappa \wedge \phi, \neg\phi) & \text{otherwise} \end{cases} \tag{6.10}$$

$$C_{\max}(\kappa, \phi) := \begin{cases} D_{\max}(\kappa \wedge \phi, \neg\phi) & \text{if } \kappa \wedge \phi \text{ is satisfiable} \\ -D_{\min}(\kappa \wedge \neg\phi, \phi) & \text{otherwise} \end{cases} \tag{6.11}$$

The meaning of previous definition is explained in 6.1, which summarises the possible combinations of values of $C(\kappa, \phi)$, as a result of the logic in 6.2 and definitions in (6.10) and (6.11). $C_{\max}$ and $C_{\min}$ can be positive, negative. A positive value indicates a state is critical w.r.t. ϕ, and the value represents how far the state is from licit state (i.e. states that does not satisfy ϕ). A negative value indicates the state is non-critical w.r.t. ϕ, and its absolute value represents how far it is from ϕ.

Algorithm 6.1 shows the pseudoalgorithm to compute the criticality $C(\kappa, \phi)$ of the current CPS state. Logical expressions κ_s and κ_t represent the expression κ where each variable is replaced with a symbol in fresh sets s and t respectively. Similarly for ϕ_s and ϕ_t. Moreover, δ is a fresh symbol that is bound in the SMT solver to the expression that represent the metric on $\mathbb{R}^n$ of choice. This enables to handle expressions $\kappa \wedge \phi$ and $\kappa \wedge \neg\phi$ easily without variable clashes.

$C_{\min}$	$C_{\max}$	Meaning
negative	negative	$\kappa \wedge \neg\phi$ sat and $\kappa \wedge \phi$ unsat. State is non critical regardless unobservables. $-C_{\min}$ and $-C_{\max}$ are the best and worst proximity values to ϕ
negative	positive	$\kappa \wedge \neg\phi$ sat and $\kappa \wedge \phi$ sat. State could be critical or not depending on unobservables. Assisted check returned for further refinement. $-C_{\min}$ is the proximity to ϕ in the best case and $C_{\max}$ is the criticality (i.e. proximity to $\neg\phi$) in the worst case.
positive	positive	$\kappa \wedge \neg\phi$ unsat and $\kappa \wedge \phi$ sat. State is critical regardless unobservables, $C_{\min}$ and $-C_{\max}$ are the worst and best criticality values (i.e. proximity to $\neg\phi$)

Table 6.1: Meaning of the results of the Reasoner.

Algorithm 6.1 Computation of proximity range.

```
Require:
  $\kappa_s, \phi_s, \kappa_t, \phi_t$: instances of $\kappa$ and $\phi$ with two distinct sets of fresh symbols
  $\delta$: fresh real symbol bound to distance expression on symbol sets $s$ and $t$
  $\epsilon$: error tolerance
  function RANKINGRANGE($\kappa$, $\phi$)
      solver ← new SMT-Optimizing-Solver
      solver.minimize-goal($\delta$).assert($\neg\phi_s \wedge \phi_t$).assert($\kappa_s$)
      model ← solver.check-sat()
      if model not found then                    ▷ $\kappa_s \wedge \neg\phi_s$ unsat: state is critical
          solver.remove($\kappa_s$).assert($\kappa_t$)
          model ← solver.check-sat()
          (Cmin, Cmax) ← (model.getvalue($\delta$), DMAX(solver))
      else                                                        ▷ $\kappa_s \wedge \neg\phi_s$ sat
          solver.remove($\kappa_s$).assert($\kappa_t$)
          model ← solver.check-sat()
          if model not found then              ▷ $\kappa_t \wedge \phi_t$ unsat: state is not critical
              (Cmin, Cmax) ← (-DMAX(solver), -model.getvalue($\delta$))
          else                                       ▷ both $\kappa \wedge \neg\phi$ and $\kappa \wedge \phi$ sat
              Cmin ← -DMAX(solver)
              solver.remove($\kappa_t$).assert($\kappa_s$)
              Cmax ← DMAX(solver)
      return (Cmin, Cmax)
  function DMAX(solver)                         ▷ iterative search for max distance
      model ← solver.get-model()
      repeat
          dmax ← model.getvalue($\delta$)                   ▷ current approximation
          solver.assert $\delta >$ dmax $+ \epsilon$
          model ← solver.check-sat()
      until model is not found
      return dmax
```

7 IMPLEMENTATIONN AND EXPERIMENTAL RESULTS

7.1 OVERVIEW

This chapter shows the experimental results to validate the proposed monitoring approach. The experimental validation is twofold: a real-time simulation of a simple use case, run in parallel to our prototype, and a set of benchmarks to evaluate the computational times. The former is aimed at verifying that the framework is a feasible monitor solution for realistic systems, the latter at understanding the limit of the approach as a function of the kind and the complexity of critical conditions.

7.2 IMPLEMENTING THE MONITORING FRAMEWORK

The observer component, which collects the values of the variables of interest, is implemented using off-the-shelf tools. The traffic network is analysed using Tshark, the command line tool of Wireshark [97, 107]. Network traffic is scanned to find Modbus commands and to extract relevant information through deep packet inspection. This way, it is possible to capture write commands sent to actuators to change their setpoints and read commands sent to sensors, with the related response containing the measurement value. This way network traffic is transformed into log format with the timestamps of observations and collected in a timeseries database.

Network traffic inspection and system logs provide passive observation of the variables of interest in ICS, since industrial protocol usually do not provide security mechanisms. The Internet of Things devices, instead, typically use telemetry protocols like MQTT [2] that provide authentication and authorisation mechanisms together with TLS tunnels. For this reason, passive network analysis is not viable in well designed real systems. Fortunatly, a data gateway can be used to this aim for authorised users.

All the collected events and measurements are stored in a timeseries database. Our prototype uses InfluxDB [63], a well established open source database which is capable of storing timestamped series of numeric values and strings.

At run-time, while the database is populated with the values of the variables of interest, the reasoner gets these values issuing one query for each variable. This way the observer and the reasoner are decoupled. The reasoner builds its knowledge from the observer and possible refinements from human operators. In our prototype both are stored in InfluxDB.

In order for the framework to be applicable to a large number of scenarios, the approach needs to be agnostic with respect to the sources of observed data and to the technique to collect them. In particular, a passive monitoring framework needs to be decoupled from the CPS by definition. Since the frequency of data collection and the computational performance of the reasoner might be different, decoupling analyses from observations is important for framework to be feasibility in real cases.

The reasoner component is made by Python code on top of two main libraries. The PuLP library is an open source linear programming modeller written in python. It can generate LP problems from a unified description and pass them to external and well established LP solvers. This way, it is not necessary to learn and use the API specific of each solvers. Our prototype uses the GLPK and the GUROBI backends. The former is open source, the latter is commercial but offers a free license for academic purposes. The comparison of the linear programming solver goes beyond the scope of this work. For this reason, our performance benchmarks take only the result of the best one as a reference.

The Microsoft Z3 Prover [33] is an open source Satisfiability Modulo Theories (SMT) solver. While other solvers are available, Z3 seems offer the best combination of state-of-the-art performance, usability of API, and documentation. Moreover, together with OptiMathSAT [99], it is the only SMT solver providing optimisation goals. The Z3 solver provides API to easily define boolean, integer, and real variable symbols and logic expression on top of such symbols. Our prototype uses the linear integer and rational arithmetics, with and without quantifiers, and with and without optimisation goals.

7.2.1 *Building Heating Example Testbed*

This section briefly shows how one of our testbed is implemented and the first experimental results that prove the feasibility of the approach.

We set up a Docker-based [38] simulation of the CPS example described in Section 4.2, and specifically the Process Control Network and the Modbus traffic between the main controller and the PLCs, with the following main containers:

- **plcsim**: our python application simulating the PLCs and the physical process. The Modbus interface is implemented using the `pymodbus` Python library [31]. Physics is simulated using the Newton's Law of Cooling, often used in literature (e.g. [28]).
- **nodered**: the HMI, the human manual operations, and the automatic operation logic of Table 4.1, and the attackers command are implemented using *Node-RED* [65], a flow-based programming tool that supports the Modbus protocol. This way licit and illicit Modbus commands are identical w.r.t their packet signature.
- **monitor**: our Python prototype of the proposed monitoring framework, which detects critical conditions and computes the proximity and proximity range from them using the SMT open source Z3 prover [34].
- **influxdb** [63] and **grafana** [52]: a time series database and a data visualisation software that provide the graphical user interface. Figure 7.1 shows a screenshot.

In this way the environment is able to simulate the main components depicted in Figure 4.1 and the Process Control Network that exhibits a real Modbus network traffic that is possible to capture and analyse. For instance, the observer component collects the value of variable T_1 by monitoring the Modbus with destination IP of PLC1, port 502, and inspects such traffic to extract from the payload the value of input register (the Modbus term indicating a read-only integer register) corresponding to T_1.

The observer uses open source tools to monitor the network and system logs, according to variable specification. In particular, the Modbus network traffic is analysed through tshark, the command line tool of Wireshark [32], to extract the values of interest through its basic deep packet inspection functionalities.

The reasoner component is a Python application that implements Figure 6.2. It iteratively gathers the observed values from the observer and, for each critical condition in the criticality specification, it uses the open source SMT Z3 solver [34] with the Python API to evaluates the criticality of the CPS state, to compute the minimal assisted check defined in (6.6), and to compute the criticality range from the critical condition as defined in Algorithm 6.1.

Figure 7.1 shows our first implementation of the graphical user interface to provide security operators with the real-time results of our framework, given a certain critical condition ϕ. The first block, in tabular form, shows different moments of the recent history. The first column shows the time of the computation of the reasoner. The third column contain the value "critical", "non critical", or "unknown" representing the three cases described in the flow chart in Figure 6.2. The forth column is empty in case of "critical" and "non critical", and contains the minimal assisted check γ. The fifth column contains the user refinement ρ, if provided.

The rest of the graphical dashboard shows the criticality range. The two gauges represent the real-time values of $C_{\min}$ and $C_{\max}$. The chart shows the timeseries values of $C_{\min}$ and $C_{\max}$. In this case, it is easy to see that the values are positive but constantly decreasing, thus the system is critical but getting closer to licit states. In the leftmost part of the chart $C_{\min}$ and $C_{\max}$ are both close to 1, hence unobservable variables have a low impact on the criticality. The central part of the chart shows that $C_{\min}$ and $C_{\max}$ greatly differ: the unobservable variables have a bigger role on the actual criticality measure of the CPS. This is the case when a refinement from the human operator can really improve the results of the reasoner. The rightmost part of the chart shows that the system is less critical, because both $C_{\min}$ and $C_{\max}$ are close to 0.

The whole prototype works on an Intel Core i7 laptop with 8 GB or RAM, and it is capable of discriminating the criticality and compute proximity ranges at real-time with an update frequency of about 500 milliseconds, which seems enough for a security monitoring solution.

While better performance tests and a characterisation of the attacks and critical conditions are subject of further investigation, first results seem to validate the overall feasibility of the approach.

7.2.2 *Chemical Process Example Testbed*

We developed a prototype to implement the monitoring components and algorithms explained in Chapters 5 and 6 to prove the feasibility of the approach.

In order for the framework to be applicable to a large number of scenarios, the approach needs to be agnostic with respect to the sources of observed data and to the techniques to collect them. The overall architecture is depicted in Figure 7.2, which shows different observation sources, possible aggregates function on observed values, a timeseries database (TSDB) to decouple the observer from the reasoner, and the main components of the reasoner that reflect Chapter 6.

The prototype is based on Docker [38] containers with a microservice architecture made of freely available open source tools and ad-hoc software developed by the author:

- **Chemical Process Simulation**: a Node-RED [65] docker container used to simulate[1]:
  1. The physical simulation of pumps and liquid flows, developed in the Typescript language.
  2. The HMI implementing the manual control of the process, developed using Typescript functions and the Node-RED visual language. 7.3 shows a screenshot.
  3. The automatic control, emulating the SCADA server, developed in Typescript and Node-RED.
  4. The attacker's read and write commands to PLCs to emulate the scenarios in Section 4.3.
- **The Observer**: Modbus-like network messages from the chemical process simulations are stored in timeseries database with

1 We developed a first version based on a Redis to simulate the physical behaviour using Lua scripts and a Python simulation of PLCs based on the Pymodbus [31] library to send real Modbus messages on the network. The Observer used TShark/Wireshark [107] for traffic capturing and the same timeseries databases to store parsed messages. The real deep packet inspection for CPS, already established in literature [88, 90], was too cumbersome for our goal since the Observer that can make use of parsed messages from the simulator without loss of generality and applicability.

all the required fields. Each variable occurring in critical specifications is associated with a query that returns one timestamped value or fails in case of unobservable variables. The timeseries DB of choice is InfluxDB [63] with its native query language and DataFrame files queries using the Pandas library.

- **The Reasoner**: the prototype of the core of the proposed framework, developed in Python using Microsoft Z3, an open source SMT prover [34]. It implements the concepts described in Chapter 6 and provides the first performance and feasibility measurements. Results are store in timeseries databases for easy access.
- **Monitoring Interface**: Grafana [52] and Chronograf (part of the InfluxData suite [63]) containers that provide mature data visualisation and query interface to time series databases.

7.3 PERFORMANCE BENCHMARK

This section shows the results of the first performance benchmark of our prototype of the framework. We show only the results for the unobservable core, which is the more complete in terms of features.

Each test generates random critical conditions based on a different number of variables up to 200. Then it generates a random CPS state. We run different set of tests with different percentages of observable values: 100%, 50%, 20%. The maximum computation time is about 4 seconds, which proves the feasibility of our framework in real cases. It is worth noticing that, while the 50% and 20% cases exhibit similar computational times, the 100% one is clearly easier to compute. This was expected, since unobservable variables require optimisation computations on wider space. Notice that the overall computational time is super-linear w.r.t. the number of variables.

We now describe a more complex benchmark, which constructs the following two formulae

$$\phi_1 = \bigwedge_{i=1}^{n} -1 \leq v_i \leq 1 \tag{7.1}$$

$$\phi_2 = \bigvee_{i=1}^{n} -v_i \leq -1 \vee 1 <= v_i \tag{7.2}$$

SMT solvers uses heuristics and linear programming techniques to achieve better results, but unlike LP solvers do not require the constraint to be conjunction of linear atoms. For this reason, we explicitly expressed ϕ_1 and ϕ_2 in conjunctive and disjunctive form to test the possible impact of their form on the performance. At each iteration, a random state s is generated such that either $s \in \mathcal{S}(\phi_1)$ or $s \in \mathcal{S}(\phi_2)$.

Our prototype is based on the following operations, using an object oriented notation:

- *new framwork(n variables)*: return a new framework with n variables
- *f.init(ϕ, metric)*: initialise the SMT solver with the critical formula ϕ and its associated metric.
- *f.compute(observation)*: compute the criticality range given a current (partial) observation.

The benchmark iteratively executes:

```
f ← new framework($n$ variables)
measure time: f.init($\phi$, metric)
for $h$ times do
    observation ← randomly generate partial observation
    measure time: result ← f.criticalityrange(observation)
```

where the Manhattan metric is used and the observation is generated in two ways:

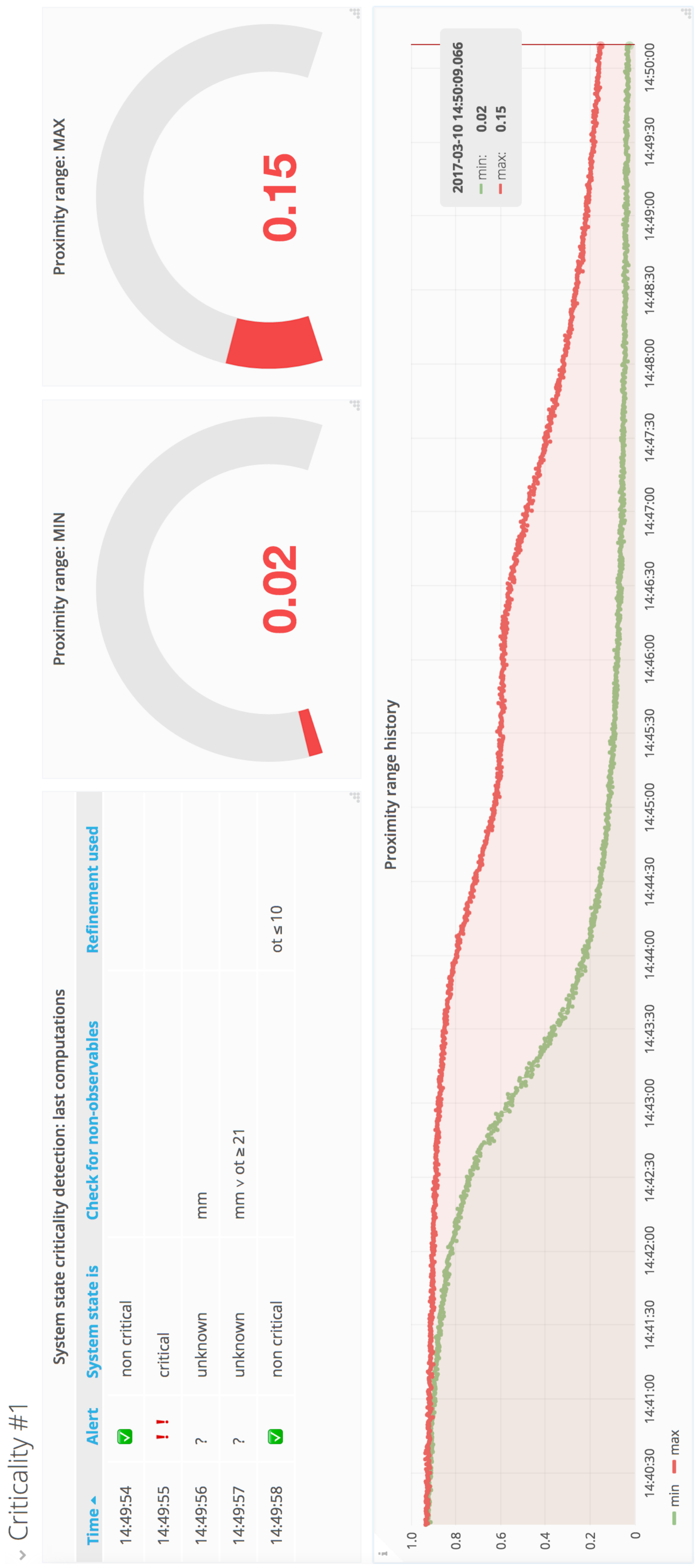
- *Outside*, i.e. a random value > 2 is generated for each variables
- *Inside*, i.e. a random value $\in [-0.5, 0.5]$ is generated for each variables.

The results are shown in three Figures. Figure 7.5 considers the case where the state is fully observable, i.e. each variable has a generated value. The left side represents the state generated as *Outside*, the right as *Inside*. The vertical bars represent the standard deviations, since for each combination of variable of inside/outside five experiments are performed. The first raw represent experiments using ϕ_1, the second using ϕ_2, and the third plots the difference between the two (where the error bar are the sum of errors.)

Notice that there is a big performance difference between the *Outside* and the *Inside*, and also a relevant difference between ϕ_1 and ϕ_2.

In comparison, Figure 7.6 and Figure 7.7 show the same results for different percentage of observables. Again, the difference is relevant. The plots also show that the initialisation time is a relevant overhead only for experiment that show a good computational time.

This test are far from complete, but show that the framework is usable for a little number of variable (about 15) regardless the complexity of the critical condition and the number of unobservable variables. On the other hand, it shows that performances highly vary depending on the complexity of ϕ, its form, and the number of unobservable. A full test to characterise the combinations of these aspects seems necessary for expressing conditions with a high number of variables.



System state criticality detection: last computations

Time ▴	Alert	System state is	Check for non-observables	Refinement used
14:49:54	✓	non critical		
14:49:55	! !	critical		
14:49:56	?	unknown	mm	
14:49:57	?	unknown	mm ∨ ot ≥ 21	
14:49:58	✓	non critical		ot ≤ 10

Figure 7.1: Screenshot of the graphical interface of the monitoring prototype.

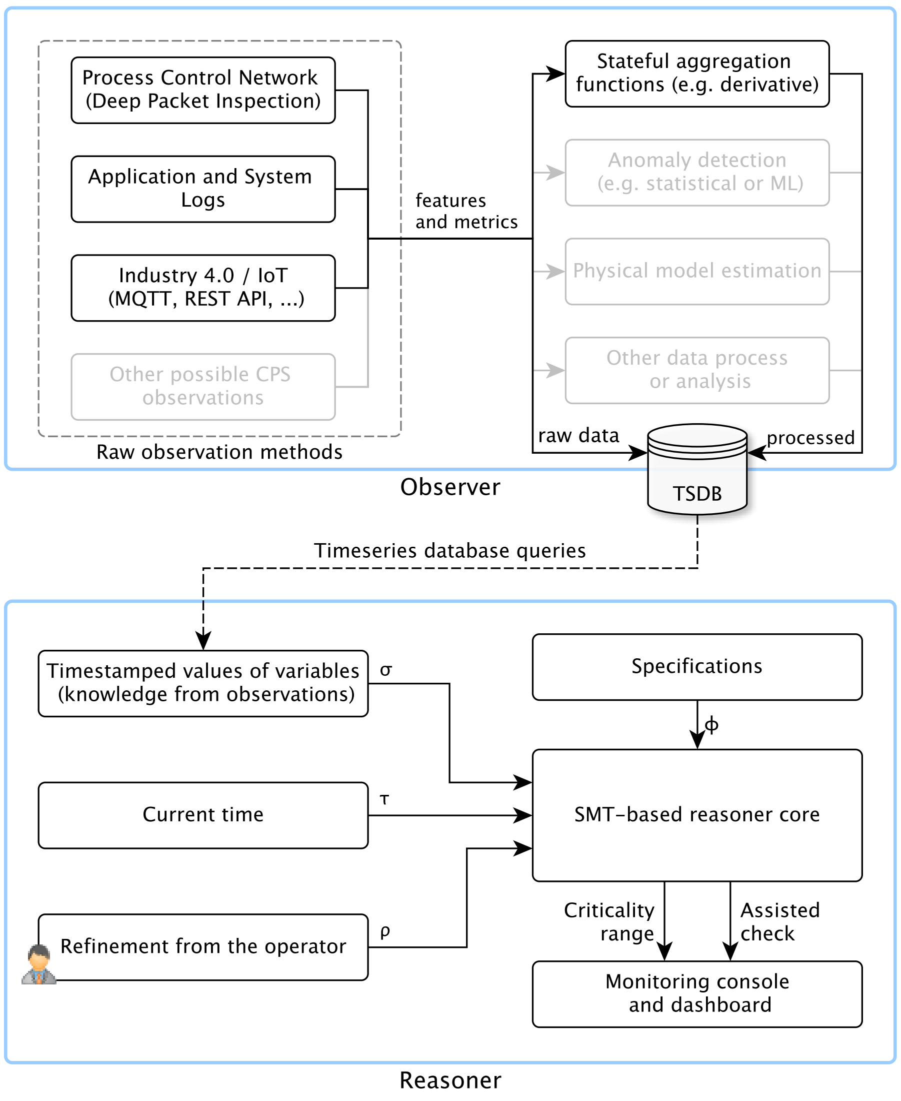

Process Control Network (Deep Packet Inspection)
Application and System Logs
Industry 4.0 / IoT (MQTT, REST API, ...)
Other possible CPS observations
Raw observation methods
features and metrics
Stateful aggregation functions (e.g. derivative)
Anomaly detection (e.g. statistical or ML)
Physical model estimation
Other data process or analysis
raw data
processed
TSDB
Observer
Timeseries database queries
Timestamped values of variables (knowledge from observations)
σ
Current time
τ
Refinement from the operator
ρ
Specifications
φ
SMT-based reasoner core
Criticality range
Assisted check
Monitoring console and dashboard
Reasoner

Figure 7.2: Overall Prototype Architecture.

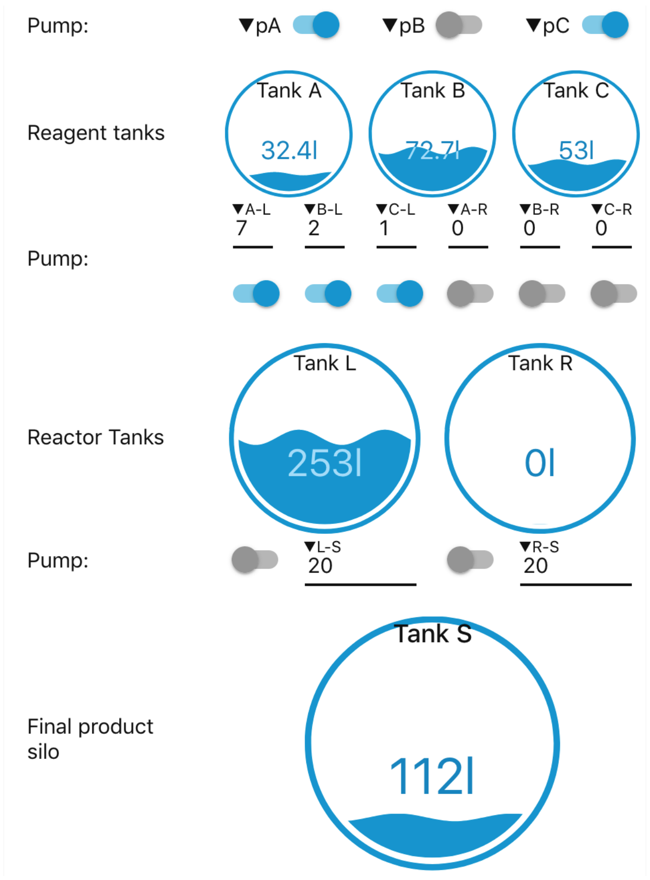


Figure 7.3: Screenshot of the emulated HMI.

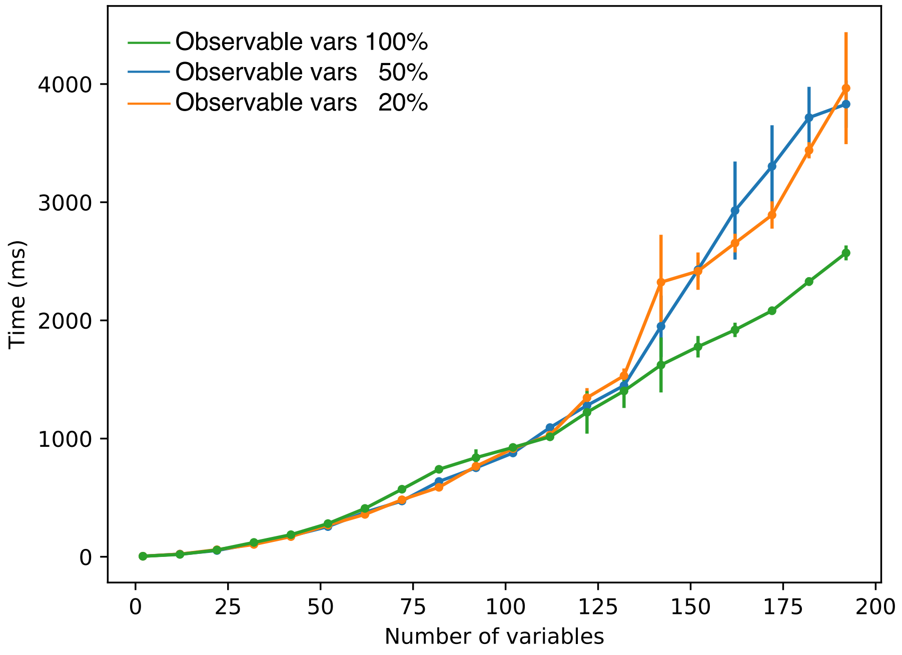


Figure 7.4: Computation time of $C(\kappa, \phi)$ from random benchmark.

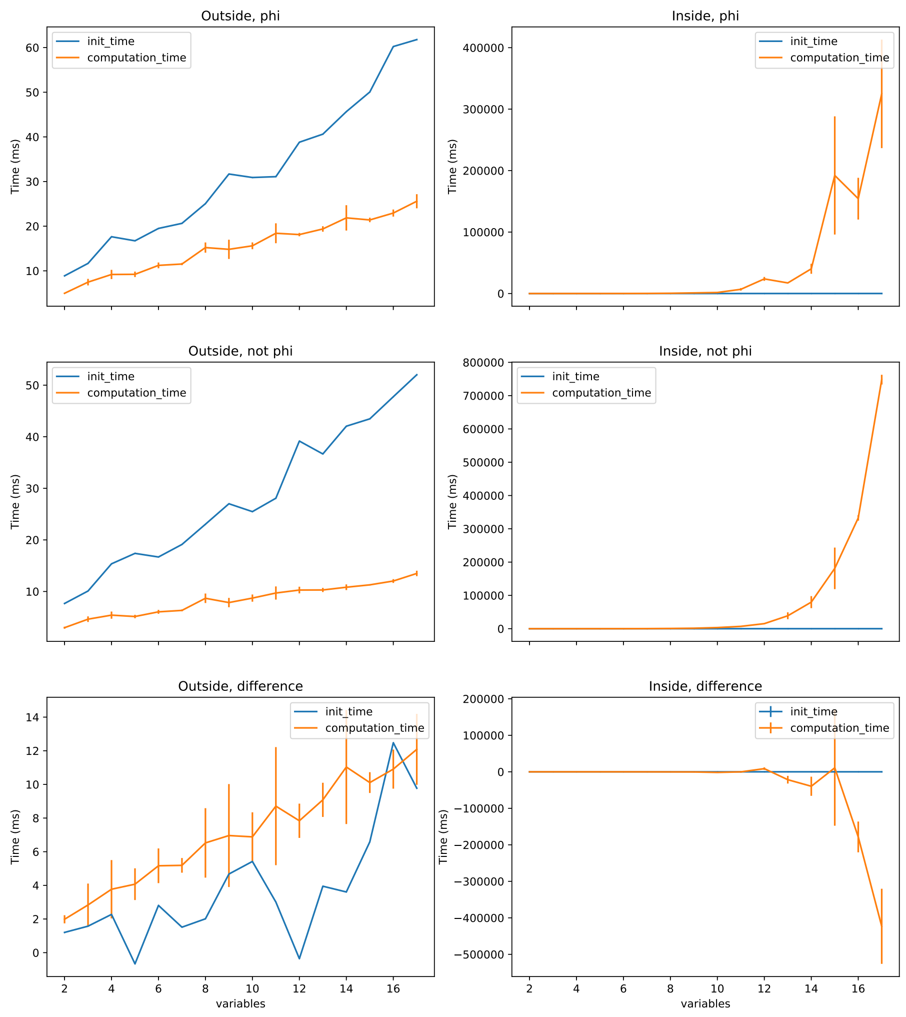


Figure 7.5: Experimental results: 100% observable.

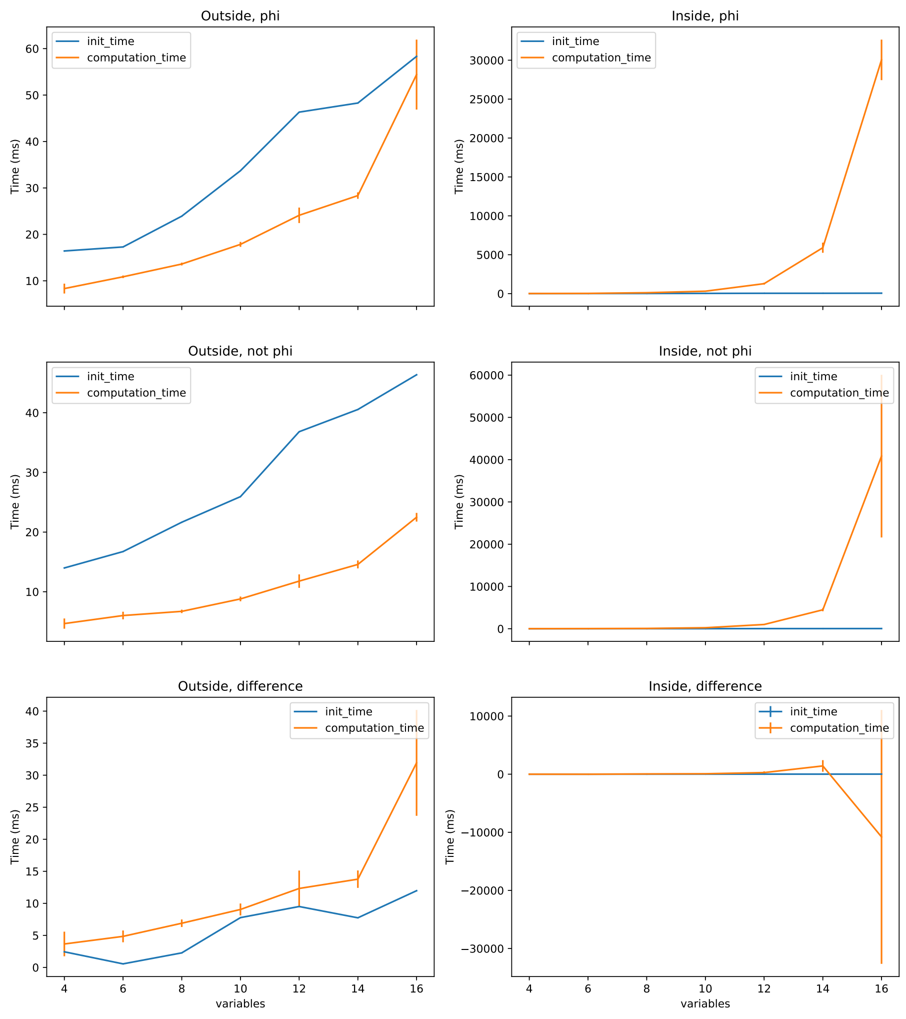


Figure 7.6: Experimental results: 50% variable observable.

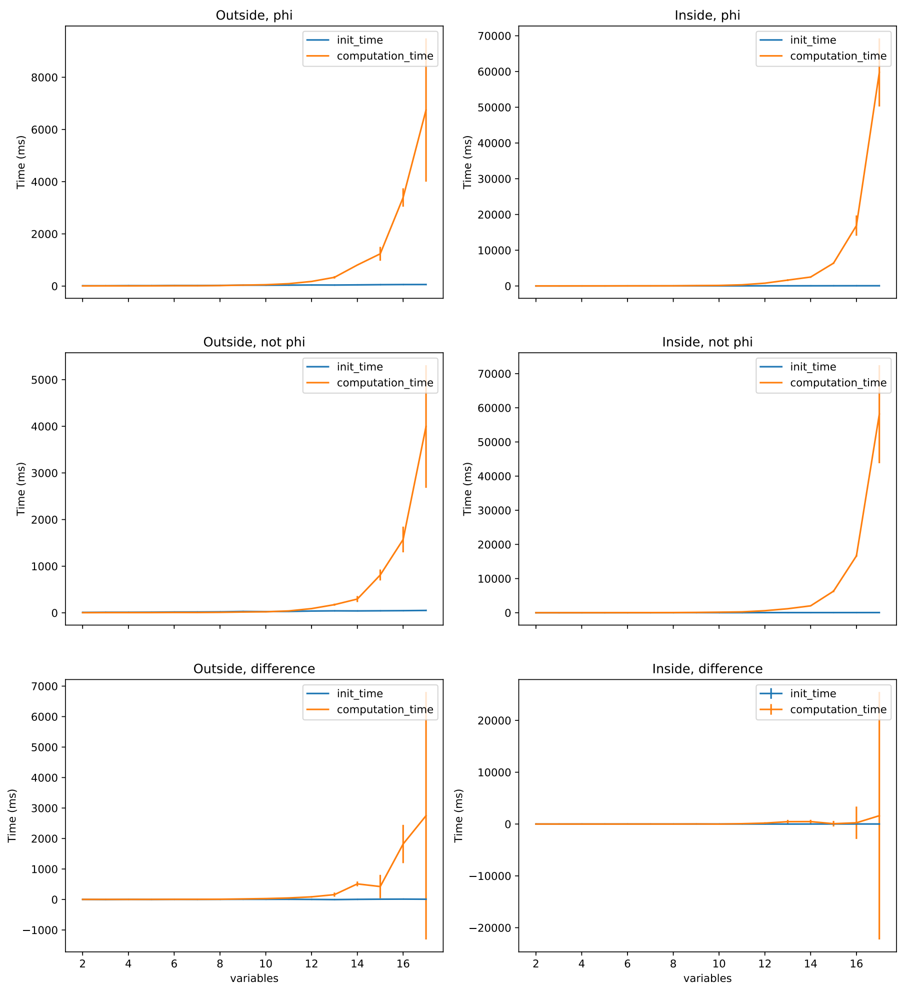


Figure 7.7: Experimental results: one observable variable.

8 CONCLUSIONS

This work aims at developing a novel and effective approach for continuous monitoring suitable for Security Operation Centres (SOC) and Computer Security Incident Response Teams (CSIRT) of enterprises that largely employ Cyber Physical System.

The first contribution is a thorough literature review. The review started from the technical aspects of the cyber components of CPS, the legacy and present technologies, their security issues, and possible solutions. This includes both academic literature, which explores the state of the art of CPS cyber security, and international standards and guidelines, aimed at fostering appropriate risk management methodologies for critical infrastructures.

My personal ten years working experience in cyber security of industrial control system has strengthened my understanding of the sector. This allowed me to carry out my industrial PhD activities within a privileged environment. The active participation in European projects, in the contest of FP7/Horizon 2020, targeting the security of the energy sector and Smart Grids, allowed me to have continuously discussions with European stakeholders.

The literature review and the working experience prove that solutions for improving the resilience of CPS are necessary. In this respect, incident response teams are considered a fundamental component of risk management and continuous monitoring and attack detection are necessary tools to support such methodologies. The literature shows that leveraging the peculiarities of CPS lead to more effectiveness techniques. Most works focus on a particular combination of observations and analysis of features, both from the cyber and the physical worlds, often presenting unsupervised or semi-supervised techniques based on statistics or machine learning. Specification-based approaches instead appear less mature.

This work presents a specification-based predictive cyber security monitoring framework for cyber physical systems. It enables specifying known critical conditions, through an easy but expressive formal language, which can be detected at run-time. It defines a notion of proximity of the CPS current state from the specified critical states: checking how the proximity changes in time enables security operators to predict whether the system is evolving towards critical states and how close it is from them. The proximity, captured by the mathematical notions of metric and premetric on $\mathbb{R}^n$, is used to define a quantitative notion of criticality of the current state.

The novelty of present work is to handle both observable and unobservable aspects of the CPS. This enables a security operator to express a model of criticality that is more complete and suitable for real cases. The monitor is able to continuously gather the value of all the observable variables from the analysis of the network traffic analysis and system logs, and to build a representation of this knowledge that correctly approximates the actual state of the system.

Unobservable variables complicate the criticality detection. When the monitor cannot discriminate if the CPS is in a critical state, a human operator can provide additional knowledge about unobservable variables as a refinement. However, this can be hard in real cases due to the complexity of the CPS and the large number of variables. To this aim, the framework is capable of computing a check in the form of a logical formula that is sufficient to discriminate the criticality of the CPS state. The formula, called assisted check, is provided to the operators as a guide for possible information refinements. The computation of the assisted check employs a method to keep the formula as small as possible, to avoid useless human effort.

Unobservable variables also complicate the computation of the criticality. However, the framework is able to compute a min/max range of the criticality which represents the best and worst case scenarios.

The recent results of Satisfiability Modulo Theories (SMT) found in literature provide the basis of the reasoning core of our framework. A contribution of this work is the development of a method to reason about and to compute the criticality of the CPS using SMT solvers. The contribution is twofold: the problem and possible computational method are formally defined, and a working prototype is presented. In case of fully observable CPS, SMT engines provide a state-of-the-art method to compute the criticality of the current state of the CPS. In presence of unobservable values, the SMT-based core provides a novel way to assist human operators for information refinements and to compute the range of the criticality of the CPS.

During the literature review and the definition and implementation of the core framework, the need for capturing temporal behaviours arose. A number of relevant works takes advantage from observing temporal patterns of the traffic captured from the process control network, as explained in Section 3.3. A contribution of this work is a literature review of online monitoring of temporal properties. It is worth noticing that the exploration of temporal specification just began and is far from being complete. Formal methods literature is a valuable reference and shows different approaches.

Model-based approaches assume that an expert defines a model of the system and a temporal property, and model checking techniques determine whether the model satisfies the property. In real CPS cases, defining a model is time consuming and often infeasible, especially for complex systems. While these techniques often allow for partial or abstract models, validating the model against the real system can be difficult. Since our monitoring solution is targeted to practices typical of security operation centres, our framework does not follow a model-based approach and only requires an operator to specify observation methods and critical conditions, i.e. only the piece of information and behaviours of interest.

Fortunately, the same formal methods can be often applied to trace semantics and to runtime monitoring. Indeed, while model checking checks every possible run of the system, monitoring checks that the exhibited behaviour complies with the specified property. Runtime verification of temporal logics provides a concise way to reason about temporal behaviours of the monitored systems. Several verification methods exist in literature. Recent works propose ad-hoc procedures for runtime monitoring of temporal logic properties. In particular recent works [3–5, 7, 9] select relevant fragments of the Metric Temporal Logic with past, future, or both operators, and provide efficient algorithms and prototypes for runtime verification. Since the SMT implementations used in this work provide programmatic API, the integration of such algorithm within our approach is allegedly possible, especially the algorithms presented in [9]. This can be the subject of future work.

Our framework is currently capable of expressing the time of the last observation of a variable and constraints on such times. Thus, the expressiveness of our timed critical specifications appear still marginal with respect to well established temporal logics. However, from the literature review, no temporal logic runtime verification method seems to be capable of continuously evaluating a notion of criticality that allows to predict if the system is evolving towards critical or licit states.

Our working prototype provides the basis for the first feasibility and performance tests. To evaluate the feasibility, two CPS examples are defined and simulated in this work, and the monitor is run in parallel. The computational performance of our framework highly depends on the size of the critical condition specifications in terms of number of variables, their forms, and the number of unobservable variables. This was expected, since the literature shows that the computational times of SMT solvers depends on a combination of algorithmic efficiency and heuristic tactics. Thus, the actual performances vary on different classes of formulae and problem and deeper performance benchmarks would be necessary to characterise different classes of formulae and to better identify strengths and limits of our approach. It is worth noticing that the optimisation modulo theory is a fast growing research area. As expected, the computational time is super-linear with respect to the complexity of the formulae. However, the first performance tests are promising and validate the proposed approach.

BIBLIOGRAPHY

[1] R. Bagnara, P. M. Hill and E. Zaffanella. 'The Parma Polyhedra Library: Toward a Complete Set of Numerical Abstractions for the Analysis and Verification of Hardware and Software Systems'. In: *Science of Computer Programming* 72.1–2 (2008), pp. 3–21.

[2] Andrew Banks and Rahul Gupta. 'MQTT Version 3.1. 1'. In: *OASIS standard* (2014).

[3] David Basin, Bhargav Nagaraja Bhatt and Dmitriy Traytel. 'Almost Event-Rate Independent Monitoring of Metric Temporal Logic'. en. In: *Tools and Algorithms for the Construction and Analysis of Systems*. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, Apr. 2017, pp. 94–112. ISBN: 978-3-662-54579-9 978-3-662-54580-5. DOI: `10.1007/978-3-662-54580-5_6`. URL: `https://link.springer.com/chapter/10.1007/978-3-662-54580-5_6` (visited on 07/12/2017).

[4] David Basin, Felix Klaedtke and Eugen Zălinescu. 'Runtime Verification of Temporal Properties over Out-of-Order Data Streams'. en. In: *Computer Aided Verification*. Lecture Notes in Computer Science. Springer, Cham, July 2017, pp. 356–376. ISBN: 978-3-319-63386-2 978-3-319-63387-9. DOI: `10.1007/978-3-319-63387-9_18`. URL: `https://link.springer.com/chapter/10.1007/978-3-319-63387-9_18` (visited on 09/01/2018).

[5] David Basin, Felix Klaedtke and Eugen Zălinescu. 'Algorithms for monitoring real-time properties'. In: *Acta informatica* 55.4 (2018), pp. 309–338.

[6] David Basin, Felix Klaedtke, Samuel Müller and Birgit Pfitzmann. 'Runtime Monitoring of Metric First-order Temporal Properties'. In: *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*. Ed. by Ramesh Hariharan, Madhavan Mukund and V. Vinay. Vol. 2. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2008, pp. 49–60. ISBN: 978-3-939897-08-8. DOI: `10.4230/LIPIcs.FSTTCS.2008.1740`. URL: `http://drops.dagstuhl.de/opus/volltexte/2008/1740` (visited on 12/09/2018).

[7] David Basin, Matúš Harvan, Felix Klaedtke and Eugen Zălinescu. 'MONPOLY: Monitoring Usage-Control Policies'. en. In: *Runtime Verification*. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, Sept. 2011, pp. 360–364. ISBN: 978-3-642-29859-2 978-3-642-29860-8. DOI: 10.1007/978-3-642-29860-8_27. URL: https://link.springer.com/chapter/10.1007/978-3-642-29860-8_27 (visited on 07/12/2017).

[8] David Basin, Felix Klaedtke, Srdjan Marinovic and Eugen Zălinescu. 'Monitoring compliance policies over incomplete and disagreeing logs'. In: *International Conference on Runtime Verification*. Springer, 2012, pp. 151–167.

[9] David Basin, Germano Caronni, Sarah Ereth, Matúš Harvan, Felix Klaedtke and Heiko Mantel. 'Scalable offline monitoring of temporal specifications'. In: *Formal Methods in System Design* 49.1-2 (2016), pp. 75–108.

[10] Marcello M. Bersani, Matteo Rossi and Pierluigi San Pietro. 'A tool for deciding the satisfiability of continuous-time metric temporal logic'. en. In: *Acta Informatica* 53.2 (Mar. 2016), pp. 171–206. ISSN: 1432-0525. DOI: 10.1007/s00236-015-0229-y. URL: https://doi.org/10.1007/s00236-015-0229-y (visited on 07/09/2018).

[11] Marcello M. Bersani, Matteo Rossi and Pierluigi San Pietro. 'A tool for deciding the satisfiability of continuous-time metric temporal logic'. In: *Acta Informatica* 53.2 (2016), pp. 171–206. ISSN: 0001-5903. DOI: 10.1007/s00236-015-0229-y. URL: http://link.springer.com/10.1007/s00236-015-0229-y.

[12] Damiano Bolzoni, Sandro Etalle, Pieter Hartel and Emmanuele Zambon. 'POSEIDON: a 2-tier Anomaly-based Network Intrusion Detection System'. In: *Fourth IEEE International Workshop on Information Assurance (IWIA)*. IEEE, 2006. ISBN: 0-7695-2564-4. DOI: 10.1109/IWIA.2006.18. URL: http://ieeexplore.ieee.org/document/1610007/.

[13] Digital Bond. *The Quickdraw Project*. 2008. URL: https://www.digitalbond.com/blog/2008/10/27/quickdraw-architecture-snort-additions/.

[14] Digital Bond. *Github: Quickdraw IDS Snort*. 2015. URL: https://github.com/digitalbond/Quickdraw-Snort.

[15] Alan Calder and Steve Watkins. *IT governance: A manager's guide to data security and ISO 27001/ISO 27002*. Kogan Page Ltd., 2008.

[16] David Carasso. *Exploring Splunk*. CITO Research, 2012.

[17] 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'. In: *IEEE Transactions on Industrial Informatics* (2011). ISSN: 1551-3203. DOI: `10.1109/TII.2010.2099234`. URL: `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5682374http://ieeexplore.ieee.org/document/5682374/`.

[18] 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'. In: *IEEE Transactions on Industrial Informatics* (2011). ISSN: 1551-3203. DOI: `10.1109/TII.2010.2099234`. URL: `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5682374http://ieeexplore.ieee.org/document/5682374/`.

[19] Andrea Carcano, Igor Nai Fovino, Marcelo Masera and Alberto Trombetta. 'State-Based Network Intrusion Detection Systems for SCADA Protocols: A Proof of Concept'. en. In: *Critical Information Infrastructures Security*. Ed. by Erich Rome and Robin Bloomfield. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, pp. 138–150. ISBN: 978-3-642-14379-3.

[20] Marco Caselli, Emmanuele Zambon and Frank Kargl. 'Sequence-aware Intrusion Detection in Industrial Control Systems'. In: *Proceedings of the 1st ACM Workshop on Cyber-Physical System Security*. CPSS '15. New York, NY, USA: ACM, 2015, pp. 13–24. ISBN: 978-1-4503-3448-8. DOI: `10.1145/2732198.2732200`. URL: `http://doi.acm.org/10.1145/2732198.2732200` (visited on 22/08/2018).

[21] Marco Caselli, Emmanuele Zambon, Jonathan Petit and Frank Kargl. 'Modeling Message Sequences for Intrusion Detection in Industrial Control Systems'. en. In: *Critical Infrastructure Protection IX*. Ed. by Mason Rice and Sujeet Shenoi. IFIP Advances in Information and Communication Technology. Springer International Publishing, 2015, pp. 49–71. ISBN: 978-3-319-26567-4.

[22] Marco Caselli, Emmanuele Zambon, Johanna Amann, Robin Sommer, Frank Kargl, Emmanuele Zambon, Johanna Amann and Frank Kargl. 'Specification Mining for Intrusion Detection in Networked Control Systems Specification Mining for Intrusion Detection in Networked Control Systems'. In: *Proceedings of the 25th USENIX Security Symposium* (2016), pp. 791–806.

[23] Brian Caswell and Jay Beale. *Snort 2.1 intrusion detection*. Syngress, 2004.

[24] Steven Cheung, Bruno Dutertre, Martin Fong, Ulf Lindqvist, Keith Skinner and Alfonso Valdes. 'Using Model-based Intrusion Detection for SCADA Networks'. In: *Science And Technology* 329 (2006), pp. 1–12. ISSN: 09598138. DOI: 10.1.1.141.2076. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.141.2076{\&}rep=rep1{\&}type=pdf.

[25] Steven Cheung, Bruno Dutertre, Martin Fong, Ulf Lindqvist, Keith Skinner and Alfonso Valdes. 'Using Model-based Intrusion Detection for SCADA Networks'. In: *Proceedings of the SCADA security scientific symposium*. Vol. 46. 2007, pp. 1–12. URL: http://www.csl.sri.com/users/cheung/SCADA-IDS-S4-2007.pdf.

[26] A. Cimatti, A. Griggio, S. Mover and S. Tonetta. 'Parameter synthesis with IC3'. In: *2013 Formal Methods in Computer-Aided Design*. Oct. 2013, pp. 165–168. DOI: 10.1109/FMCAD.2013.6679406.

[27] E. Colbert, D. Sullivan, S. Hutchinson, K. Renard and S. Smith. 'A Process-Oriented Intrusion Detection Method for Industrial Control Systems'. en. In: *Proceedings of the 2016 international conference on cyber warfare and security*. Boston, MA, Mar. 2016. URL: https://search.proquest.com/openview/ef098ce0b5dae809d64d2957f069011?pq-origsite=gscholar&cbl=396500 (visited on 22/08/2018).

[28] Alexander Cole, Benjamin Jury and Kei Takashina. 'A Leidenfrost Thermostat'. In: *Journal of Heat Transfer* 137.3 (2015), p. 034502. ISSN: 0022-1481. DOI: 10.1115/1.4029238. URL: http://heattransfer.asmedigitalcollection.asme.org/article.aspx?doi=10.1115/1.4029238.

[29] Alessio Coletta and Alessandro Armando. 'Security Monitoring for Industrial Control Systems'. In: *Security of Industrial Control Systems and Cyber Physical Systems. CyberICS 2015*. Springer International Publishing, 2016, pp. 48–62. DOI: 10.1007/978-

3-319-40385-4_4. URL: http://link.springer.com/10.1007/978-3-319-40385-4_4.

[30] Alessio Coletta and Alessandro Armando. 'Security Monitoring for Industrial Control Systems'. In: *Security of Industrial Control Systems and Cyber Physical Systems. CyberICS 2015*. Springer International Publishing, 2016, pp. 48–62. DOI: 10.1007/978-3-319-40385-4_4. URL: http://link.springer.com/10.1007/978-3-319-40385-4{_}4.

[31] Galen Collins. *pymodbus 1.2.0*. 2017. URL: https://pypi.python.org/pypi/pymodbus/.

[32] Gerald Combs and Others. 'Wireshark'. In: *www.wireshark.org* (2017).

[33] Leonardo De Moura and Nikolaj Bjørner. 'Z3: An efficient SMT solver'. In: *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340. DOI: 10.1007/978-3-540-78800-3_24.

[34] Leonardo De Moura and Nikolaj Bjørner. 'Z3: An efficient SMT solver'. In: *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer. 2008, pp. 337–340. DOI: 10.1007/978-3-540-78800-3_24.

[35] Normann Decker, Martin Leucker and Daniel Thoma. 'Monitoring Modulo Theories'. In: *Tools and Algorithms for the Construction ...* (2014), pp. 1–15. URL: http://link.springer.com/chapter/10.1007/978-3-642-54862-8{_}23.

[36] Jyotirmoy V. Deshmukh, Alexandre Donzé, Shromona Ghosh, Xiaoqing Jin, Garvit Juniwal and Sanjit A. Seshia. 'Robust online monitoring of signal temporal logic'. In: *Formal Methods in System Design* 51.1 (2017), pp. 5–30. ISSN: 15728102. DOI: 10.1007/s10703-017-0286-7. arXiv: 1506.08234.

[37] Josh Diakun, Paul R Johnson and Derek Mock. *Splunk Operational Intelligence Cookbook*. Packt Publishing Ltd, 2016.

[38] Docker Inc. *Docker*. 2017. URL: www.docker.com.

[39] Adel Dokhanchi, Bardh Hoxha and Georgios Fainekos. 'On-Line Monitoring for Temporal Logic Robustness'. In: *arXiv:1408.0045 [cs]* (July 2014). arXiv: 1408.0045. URL: http://arxiv.org/abs/1408.0045 (visited on 06/09/2018).

[40] Rui-Hong Dong, Dong-Fang Wu, Qiu-Yu Zhang and Tao Zhang. 'Traffic Characteristic Map-based Intrusion Detection Model for Industrial Internet'. en. In: *International Journal of Network Security* 20.2 (Mar. 2018), pp. 359–370. ISSN: 1816-353X. DOI: 10.6633/IJNS.201803.20(2).17.

[41] Alexandre Donzé, Thomas Ferrère and Oded Maler. 'Efficient robust monitoring for STL'. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 8044 LNCS. 2013, pp. 264–279. ISBN: 9783642397981. DOI: 10.1007/978-3-642-39799-8_19.

[42] Samuel East, Jonathan Butts, Mauricio Papa and Sujeet Shenoi. 'A Taxonomy of Attacks on the DNP3 Protocol'. In: *Critical Infrastructure Protection III*. 2009, pp. 67–81. DOI: 10.1007/978-3-642-04798-5_5. URL: http://link.springer.com/10.1007/978-3-642-04798-5{_}5.

[43] D. Evans. *The Internet of Things How the Next Evolution of the Internet Is Changing Everything*. Tech. rep. 2011. URL: http://refhub.elsevier.com/S1389-1286(18)30120-8/sbref0040.

[44] Georgios E. Fainekos and George J. Pappas. 'Robustness of temporal logic specifications for continuous-time signals'. In: *Theoretical Computer Science* 410.42 (2009), pp. 4262–4291. ISSN: 03043975. DOI: 10.1016/j.tcs.2009.06.021. URL: http://dx.doi.org/10.1016/j.tcs.2009.06.021http://linkinghub.elsevier.com/retrieve/pii/S0304397509004149.

[45] Benedikt Ferling, Justyna Chromik, Marco Caselli and Anne Remke. 'Intrusion Detection for Sequence-Based Attacks with Reduced Traffic Models'. en. In: *Measurement, Modelling and Evaluation of Computing Systems*. Ed. by Reinhard German, Kai-Steffen Hielscher and Udo R. Krieger. Lecture Notes in Computer Science. Springer International Publishing, 2018, pp. 53–67. ISBN: 978-3-319-74947-1.

[46] Igor Nai Fovino, Andrea Carcano, Marcelo Masera and Alberto Trombetta. 'Design and Implementation of a Secure Modbus Protocol'. In: *International Conference on Critical Infrastructure Protection*. Springer, 2009, pp. 83–96. DOI: 10.1007/978-3-642-04798-5_6. URL: http://link.springer.com/10.1007/978-3-642-04798-5{_}6.

[47] Wei Gao and Thomas Morris. 'On Cyber Attacks and Signature Based Intrusion Detection for Modbus Based Industrial Control Systems'. In: *Journal of Digital Forensics, Security and Law* 9.1 (Jan. 2014). ISSN: (Print) 1558-7215. DOI: `10.15394/jdfsl.2014.1162`. URL: `https://commons.erau.edu/jdfsl/vol9/iss1/3`.

[48] Wei Gao, Thomas Morris, Bradley Reaves and Drew Richey. 'On SCADA control system command and response injection and intrusion detection'. en. In: *2010 eCrime Researchers Summit*. Dallas, USA: IEEE, 2010, pp. 1–9. ISBN: 978-1-4244-7760-9 978-1-4244-7761-6. DOI: `10.1109/ecrime.2010.5706699`. URL: `http://ieeexplore.ieee.org/document/5706699/` (visited on 22/08/2018).

[49] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández and E. Vázquez. 'Anomaly-based network intrusion detection: Techniques, systems and challenges'. In: *Computers & Security* 28.1-2 (2009), pp. 18–28. ISSN: 01674048. DOI: `10.1016/j.cose.2008.08.003`. URL: `http://linkinghub.elsevier.com/retrieve/pii/S0167404808000692`.

[50] Grant Gilchrist. 'Secure authentication for DNP3'. In: *IEEE Power and Energy Society General Meeting - Conversion and Delivery of Electrical Energy in the 21st Century*. IEEE, 2008, pp. 1–3. ISBN: 978-1-4244-1905-0. DOI: `10.1109/PES.2008.4596147`. URL: `http://ieeexplore.ieee.org/document/4596147/`.

[51] Clinton Gormley and Zachary Tong. *Elasticsearch: the Definitive Guide*. O'Reilly Media, Inc., 2015.

[52] Grafana Labs. *Grafana*. 2017. URL: `https://grafana.com`.

[53] Yuvraj Gupta. *Kibana Essentials*. Packt Publishing Ltd, 2015.

[54] H. Hadeli, R. Schierholz, M. Braendle and C. Tuduce. 'Leveraging determinism in industrial control systems for advanced anomaly detection and reliable security configuration'. In: *2009 IEEE Conference on Emerging Technologies Factory Automation*. Sept. 2009, pp. 1–8. DOI: `10.1109/ETFA.2009.5347134`.

[55] Mohammad Hamad, Zain A. H. Hammadeh, Selma Saidi, Vassilis Prevelakis and Rolf Ernst. 'Prediction of abnormal temporal behavior in real-time systems'. In: *Proceedings of the 33rd Annual ACM Symposium on Applied Computing - SAC '18*. New York, New York, USA: ACM Press, 2018, pp. 359–367. ISBN:

9781450351911. DOI: 10.1145/3167132.3167172. URL: http://dl.acm.org/citation.cfm?doid=3167132.3167172.

[56] Christine Hang, Panagiotis Manolios and Vasilis Papavasileiou. 'Synthesizing Cyber-Physical Architectural Models with Real-Time Constraints'. en. In: *Computer Aided Verification*. Ed. by Ganesh Gopalakrishnan and Shaz Qadeer. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, pp. 441–456. ISBN: 978-3-642-22110-1.

[57] W Heimerdinger, V Guralnik and R VanRiper. *Anomaly-based intrusion detection*. 2006. URL: https://www.google.com/patents/US20060034305.

[58] Hsi-Ming Ho, Joël Ouaknine and James Worrell. 'On the Expressiveness and Monitoring of Metric Temporal Logic'. In: *arXiv:1803.02653 [cs]* (Mar. 2018). arXiv: 1803.02653. URL: http://arxiv.org/abs/1803.02653 (visited on 13/03/2018).

[59] J. Hong, C. Liu and M. Govindarasu. 'Detection of cyber intrusions using network-based multicast messages for substation automation'. In: *ISGT 2014*. Feb. 2014, pp. 1–5. DOI: 10.1109/ISGT.2014.6816375.

[60] Chongyuan Hou, Hanhong Jiang, Wanzhi Rui and Liang Liu. 'A Probabilistic Principal Component Analysis Approach for Detecting Traffic Anomaly in Industrial Networks [J]'. In: *Journal of Xi'an Jiaotong University* 2 (2012), p. 011. URL: http://en.cnki.com.cn/Article_en/CJFDTotal-XAJT201202011.htm.

[61] Peter Huitsing, Rodrigo Chandia, Mauricio Papa and Sujeet Shenoi. 'Attack taxonomies for the Modbus protocols'. In: *International Journal of Critical Infrastructure Protection* 1 (2008), pp. 37–44. ISSN: 18745482. DOI: 10.1016/j.ijcip.2008.08.003. URL: http://www.sciencedirect.com/science/article/pii/S187454820800005Xhttp://linkinghub.elsevier.com/retrieve/pii/S187454820800005X.

[62] J. Hurley, A. Munoz and S. Sezer. 'ITACA: Flexible, Scalable Network Analysis'. In: *Proc. IEEE Int'l Conf. on Communications Industry Forum & Exhibit*. 2012, pp. 1084–1088.

[63] InfluxData Inc. *InfluxDB*. 2017. URL: www.influxdata.com.

[64] *ISO/IEC 19464:2014 - Information technology – Advanced Message Queuing Protocol (AMQP) v1.0 specification*. URL: https://www.iso.org/standard/64955.html (visited on 06/09/2018).

[65] JS Foundation. *Node-RED*. 2017. URL: `http://nodered.org`.

[66] I. Kiss, B. Genge and P. Haller. 'A clustering-based approach to detect cyber attacks in process control systems'. In: *2015 IEEE 13th International Conference on Industrial Informatics (INDIN)*. July 2015, pp. 142–148. DOI: `10.1109/INDIN.2015.7281725`.

[67] Istvan Kiss, Bela Genge, Piroska Haller and Gheorghe Sebestyen. 'Data clustering-based anomaly detection in industrial control systems'. en. In: *2014 IEEE 10th International Conference on Intelligent Computer Communication and Processing (ICCP)*. Cluj Napoca, Romania: IEEE, Sept. 2014, pp. 275–281. ISBN: 978-1-4799-6569-4 978-1-4799-6568-7. DOI: `10.1109/ICCP.2014.6937009`. URL: `http://ieeexplore.ieee.org/document/6937009/` (visited on 22/08/2018).

[68] Marina Krotofil, Jason Larsen and Dieter Gollmann. 'The Process Matters: Ensuring Data Veracity in Cyber-Physical Systems'. In: *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*. ASIA CCS '15. New York, NY, USA: ACM, 2015, pp. 133–144. ISBN: 978-1-4503-3245-3. DOI: `10.1145/2714576.2714599`. URL: `http://doi.acm.org/10.1145/2714576.2714599` (visited on 22/08/2018).

[69] Hui Lin, Adam Slagell, Catello Di Martino, Zbigniew Kalbarczyk and Ravishankar K. Iyer. 'Adapting Bro into SCADA: Building a Specification-based Intrusion Detection System for the DNP3 Protocol'. In: *Proceedings of the Eighth Annual Cyber Security and Information Intelligence Research Workshop*. CSIIRW '13. New York, NY, USA: ACM, 2013, 5:1–5:4. ISBN: 978-1-4503-1687-3. DOI: `10.1145/2459976.2459982`. URL: `http://doi.acm.org/10.1145/2459976.2459982` (visited on 22/08/2018).

[70] Hui Lin, Adam Slagell, Zbigniew Kalbarczyk, Peter W. Sauer and Ravishankar K. Iyer. 'Semantic Security Analysis of SCADA Networks to Detect Malicious Control Commands in Power Grids'. In: *Proceedings of the First ACM Workshop on Smart Energy Grid Security*. SEGS '13. New York, NY, USA: ACM, 2013, pp. 29–34. ISBN: 978-1-4503-2492-2. DOI: `10.1145/2516930.2516947`. URL: `http://doi.acm.org/10.1145/2516930.2516947` (visited on 22/08/2018).

[71] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang and W. Zhao. 'A Survey on Internet of Things: Architecture, Enabling Technologies, Security and Privacy, and Applications'. In: *IEEE Internet*

of Things Journal 4.5 (Oct. 2017), pp. 1125–1142. ISSN: 2327-4662. DOI: 10.1109/JIOT.2017.2683200.

[72] O. Linda, M. Manic and T. Vollmer. 'Improving cyber-security of smart grid systems via anomaly detection and linguistic domain knowledge'. In: *2012 5th International Symposium on Resilient Control Systems*. Aug. 2012, pp. 48–54. DOI: 10.1109/ISRCS.2012.6309292.

[73] O. Linda, T. Vollmer and M. Manic. 'Neural Network based Intrusion Detection System for critical infrastructures'. In: *2009 International Joint Conference on Neural Networks*. June 2009, pp. 1827–1834. DOI: 10.1109/IJCNN.2009.5178592.

[74] O. Linda, M. Manic, T. Vollmer and J. Wright. 'Fuzzy logic based anomaly detection for embedded network security cyber sensor'. In: *2011 IEEE Symposium on Computational Intelligence in Cyber Security (CICS)*. Apr. 2011, pp. 202–209. DOI: 10.1109/CICYBS.2011.5949392.

[75] O. Linda, M. Manic, J. Alves-Foss and T. Vollmer. 'Towards resilient critical infrastructures: Application of Type-2 Fuzzy Logic in embedded network security cyber sensor'. In: *2011 4th International Symposium on Resilient Control Systems*. Aug. 2011, pp. 26–32. DOI: 10.1109/ISRCS.2011.6016083.

[76] LogRhythm Inc. *LogRhythm security intelligence and analytics platform*. 2017. URL: https://logrhythm.com.

[77] Y. Luo. 'Research and design on intrusion detection methods for industrial control system'. PhD. Hangzhou, China: Zhejiang University, 2013.

[78] L. A. Maglaras and J. Jiang. 'Intrusion detection in SCADA systems using machine learning techniques'. In: *2014 Science and Information Conference*. Aug. 2014, pp. 626–631. DOI: 10.1109/SAI.2014.6918252.

[79] R. Mahmoud, T. Yousuf, F. Aloul and I. Zualkernan. 'Internet of things (IoT) security: Current status, challenges and prospective measures'. In: *2015 10th International Conference for Internet Technology and Secured Transactions (ICITST)*. Dec. 2015, pp. 336–341. DOI: 10.1109/ICITST.2015.7412116.

[80] Munir Majdalawieh, Francesco Parisi-Presicce and Duminda Wijesekera. 'DNPSec: Distributed network protocol version 3 (DNP3) security framework'. In: *Advances in Computer, Informa-*

tion, and Systems Sciences, and Engineering. Springer, 2007, pp. 227–234.

[81] Robert Mitchell and Ing Ray Chen. 'Behavior Rule Specification-Based Intrusion Detection for Safety Critical Medical Cyber Physical Systems'. In: *IEEE Transactions on Dependable and Secure Computing* 12.1 (2015), pp. 16–30. ISSN: 15455971. DOI: 10.1109/TDSC.2014.2312327.

[82] Stefan Mitsch and André Platzer. 'ModelPlex: verified runtime validation of verified cyber-physical system models'. en. In: *Formal Methods in System Design* 49.1 (Oct. 2016), pp. 33–74. ISSN: 1572-8102. DOI: 10.1007/s10703-016-0241-z. URL: https://doi.org/10.1007/s10703-016-0241-z (visited on 11/09/2018).

[83] Y. Mo, R. Chabukswar and B. Sinopoli. 'Detecting Integrity Attacks on SCADA Systems'. In: *IEEE Transactions on Control Systems Technology* 22.4 (July 2014), pp. 1396–1407. ISSN: 1063-6536. DOI: 10.1109/TCST.2013.2280899.

[84] *MODBUS Application Protocol Specification V1.1b3*. Tech. rep. 2012, pp. 1–50. URL: http://www.modbus.org.

[85] T. H. Morris, B. A. Jones, R. B. Vaughn and Y. S. Dandass. 'Deterministic Intrusion Detection Rules for MODBUS Protocols'. In: *2013 46th Hawaii International Conference on System Sciences*. Jan. 2013, pp. 1773–1781. DOI: 10.1109/HICSS.2013.174.

[86] T. Morris, R. Vaughn and Y. Dandass. 'A Retrofit Network Intrusion Detection System for MODBUS RTU and ASCII Industrial Control Systems'. In: *2012 45th Hawaii International Conference on System Sciences*. Jan. 2012, pp. 2338–2345. DOI: 10.1109/HICSS.2012.78.

[87] Christian Moya, Junho Hong and Jiankang Wang. 'Application of Correlation Indices on Intrusion Detection Systems: Protecting the Power Grid Against Coordinated Attacks'. In: *arXiv:1806.03544 [cs]* (June 2018). arXiv: 1806.03544. URL: http://arxiv.org/abs/1806.03544 (visited on 22/08/2018).

[88] Igor Nai Fovino, Andrea Carcano, Alessio Coletta, Michele Guglielmi, Marcelo Masera and Alberto Trombetta. 'State-Based Firewall for Industrial Protocols with Critical-State Prediction Monitor'. In: *Critical Information Infrastructures Security*. Vol. 6712 LNCS. 2011, pp. 116–127. ISBN: 9783642216930. DOI: 10.1007/

978-3-642-21694-7_10. URL: http://link.springer.com/10.1007/978-3-642-21694-7{_}10.

[89] Igor Nai Fovino, Alessio Coletta, Andrea Carcano and Marcelo Masera. 'Critical State-Based Filtering System for Securing SCADA Network Protocols'. In: *IEEE Transactions on Industrial Electronics* 59.10 (Oct. 2012), pp. 3943–3950. ISSN: 0278-0046. DOI: 10.1109/TIE.2011.2181132. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6111289http://ieeexplore.ieee.org/document/6111289/.

[90] Igor Nai Fovino, Alessio Coletta, Andrea Carcano and Marcelo Masera. 'Critical State-Based Filtering System for Securing SCADA Network Protocols'. In: *IEEE Transactions on Industrial Electronics* 59.10 (2012), pp. 3943–3950. ISSN: 0278-0046. DOI: 10.1109/TIE.2011.2181132. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6111289http://ieeexplore.ieee.org/document/6111289/.

[91] Angela Orebaugh, Gilbert Ramirez and Jay Beale. *Wireshark & Ethereal network protocol analyzer toolkit*. Syngress, 2006.

[92] Vern Paxson. 'Bro: a system for detecting network intruders in real-time'. In: *Computer Networks* 31.23 (Dec. 1999), pp. 2435–2463. ISSN: 1389-1286. DOI: 10.1016/S1389-1286(99)00112-7. URL: http://www.sciencedirect.com/science/article/pii/S1389128699001127 (visited on 22/08/2018).

[93] Vern Paxson. 'Bro: a system for detecting network intruders in real-time'. In: *Computer Networks* 31.23-24 (1999), pp. 2435–2463. ISSN: 13891286. DOI: 10.1016/S1389-1286(99)00112-7. URL: http://linkinghub.elsevier.com/retrieve/pii/S1389128699001127.

[94] M Roesch. 'Snort: Lightweight Intrusion Detection for Networks.' In: *LISA '99: 13th Systems Administration Conference* (1999), pp. 229–238. URL: http://static.usenix.org/publications/library/proceedings/lisa99/full{_}papers/roesch/roesch.pdf.

[95] Gurpreet S Sachdeva. *Practical ELK Stack*. Apress, 2017.

[96] P. Saint-Andre. 'Extensible Messaging and Presence Protocol (XMPP): Core'. In: (2011). ISSN: 2070-1721. URL: http://www.rfc-editor.org/info/rfc6120 (visited on 06/09/2018).

[97] Chris Sanders. *Practical packet analysis: Using Wireshark to solve real-world network problems*. No Starch Press, 2011.

[98] E. Eugene Schultz. 'Continuous Monitoring: What It Is, Why It Is Needed, and How to Use It'. en. In: *Whitpaper - SANS Institute InfoSec Reading Room* (June 2011), p. 16.

[99] Roberto Sebastiani and Patrick Trentin. 'OptiMathSAT: A Tool for Optimization Modulo Theories'. In: *International Conference on Computer Aided Verification*. Springer, 2015, pp. 447–454. DOI: 10.1007/978-3-319-21690-4_27.

[100] Yasser Shoukry, Michelle Chong, Masashi Wakaiki, Pierluigi Nuzzo, Alberto Sangiovanni-Vincentelli, Sanjit A. Seshia, JOÃo P. Hespanha and Paulo Tabuada. 'SMT-Based Observer Design for Cyber-Physical Systems Under Sensor Attacks'. In: *ACM Trans. Cyber-Phys. Syst.* 2.1 (Jan. 2018), 5:1–5:27. ISSN: 2378-962X. DOI: 10.1145/3078621. URL: http://doi.acm.org/10.1145/3078621 (visited on 11/09/2018).

[101] Peter Stavroulakis and Mark Stamp. *Handbook of Information and Communication Security*. en. Springer Science & Business Media, Feb. 2010. ISBN: 978-3-642-04117-4.

[102] Keith Stouffer, Victoria Pillitteri, Suzanne Lightman, Marshall Abrams and Adam Hahn. *Guide to Industrial Control Systems (ICS) Security*. Tech. rep. Gaithersburg, MD: National Institute of Standards and Technology, 2015. DOI: 10.6028/NIST.SP.800-82r2. URL: http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-82r2.pdf.

[103] Suricata. *Suricata Open Source IDS / IPS / NSM engine*. 2017. URL: http://suricata-ids.org.

[104] Chi-Ho Tsang and S. Kwong. 'Multi-agent intrusion detection system in industrial network using ant colony clustering approach and unsupervised feature extraction'. In: *2005 IEEE International Conference on Industrial Technology*. Dec. 2005, pp. 51–56. DOI: 10.1109/ICIT.2005.1600609.

[105] James Turnbull. *The Logstash Book*. James Turnbull, 2013.

[106] T. Vollmer and M. Manic. 'Computationally efficient Neural Network Intrusion Security Awareness'. In: *2009 2nd International Symposium on Resilient Control Systems*. Aug. 2009, pp. 25–30. DOI: 10.1109/ISRCS.2009.5251357.

[107] Wireshark Foundation. *Wireshark*. 2017. URL: www.wireshark.org.

[108] Kun Xiao, Nianen Chen, Shangping Ren, Limin Shen, Xianhe Sun, Kevin Kwiat and Michael Macalik. 'A Workflow-Based Non-intrusive Approach for Enhancing the Survivability of Critical Infrastructures in Cyber Environment'. In: *Third International Workshop on Software Engineering for Secure Systems (SESS'07: ICSE Workshops 2007)*. IEEE, 2007, pp. 4–4. ISBN: 0-7695-2952-6. DOI: 10.1109/SESS.2007.3. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4273330http://ieeexplore.ieee.org/document/4273330/.

[109] X. Yang, J. Lin, W. Yu, P. Moulema, X. Fu and W. Zhao. 'A Novel En-Route Filtering Scheme Against False Data Injection Attacks in Cyber-Physical Networked Systems'. In: *IEEE Transactions on Computers* 64.1 (Jan. 2015), pp. 4–18. ISSN: 0018-9340. DOI: 10.1109/TC.2013.177.

[110] Y. Yang, K. McLaughlin, T. Littler, S. Sezer, B. Pranggono and H. F. Wang. 'Intrusion Detection System for IEC 60870-5-104 based SCADA networks'. In: *2013 IEEE Power Energy Society General Meeting*. July 2013, pp. 1–5. DOI: 10.1109/PESMG.2013.6672100.

[111] Y. Yang, K. McLaughlin, T. Littler, S. Sezer and H. F. Wang. 'Rule-based intrusion detection system for SCADA networks'. In: *2nd IET Renewable Power Generation Conference (RPG 2013)*. Sept. 2013, pp. 1–4. DOI: 10.1049/cp.2013.1729.

[112] W. Yusheng, F. Kefeng, L. Yingxu, L. Zenghui, Z. Ruikang, Y. Xiangzhen and L. Lin. 'Intrusion Detection of Industrial Control System Based on Modbus TCP Protocol'. In: *2017 IEEE 13th International Symposium on Autonomous Decentralized System (ISADS)*. Mar. 2017, pp. 156–162. DOI: 10.1109/ISADS.2017.29.

[113] Min Zhang and Yunhui Ying. 'Towards SMT-based LTL Model Checking of Clock Constraint Specification Language for Real-time and Embedded Systems'. In: *Proceedings of the 18th ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems*. LCTES 2017. New York, NY, USA: ACM, 2017, pp. 61–70. ISBN: 978-1-4503-5030-3. DOI: 10.1145/3078633.3081035. URL: http://doi.acm.org/10.1145/3078633.3081035 (visited on 07/09/2018).

[114] K. Zhao and L. Ge. 'A Survey on the Internet of Things Security'. In: *2013 Ninth International Conference on Computational*

Intelligence and Security. Dec. 2013, pp. 663–667. DOI: 10.1109/CIS.2013.145.

[115] Christopher Zimmer, Balasubramanya Bhat, Frank Mueller and Sibin Mohan. 'Time-based intrusion detection in cyber-physical systems'. In: *Proceedings of the 1st ACM/IEEE International Conference on Cyber-Physical Systems - ICCPS '10* (2010), p. 109. DOI: 10.1145/1795194.1795210. URL: http://portal.acm.org/citation.cfm?doid=1795194.1795210.