PREDICTIVE DETECTION OF KNOWN SECURITY CRITICALITIES IN CYBER PHYSICAL SYSTEMS WITH UNOBSERVABLE VARIABLES

Alessio Coletta[1,2]

[1]Security and Trust Unit, Bruno Kessler Foundation, Trento, Italy
[2]Department of Information Engineering and Computer Science,
University of Trento, Italy

ABSTRACT

A large number of existing Cyber Physical Systems (CPS) in production environments, also employed in critical infrastructures, are severely vulnerable to cyber threats but cannot be modified due to strict availability requirements and nearly impossible change management. Monitoring solutions are increasingly proving to be very effective in such scenarios. Since CPS are typically designed for a precise purpose, their behaviour is predictable to a good extent and often well known, both from the process and the cyber perspective. This work presents a cyber security monitor capable of leveraging such knowledge to detect illicit activities. It uses a formal language to specify critical conditions and an SMT-based engine to detect them through network traffic and log analysis. The framework is predictive, i.e. it recognises if the system is approaching a critical state before reaching it. An important novelty of the approach is the capability of dealing with unobservable variables, making the framework much more feasible in real cases. This work presents the formal framework and first experimental results validating the feasibility of the approach.

KEYWORDS

Security Monitoring, Detection and Prevention Systems, Critical Infrastructures, Cyber Physical Systems, SMT.

1. INTRODUCTION

Cyber Physical Systems (CPS) are composed by networked ICT devices that support the operation of physical entities. In this work we use CPS as general term that includes Industrial Control Systems (ICS), building automation systems, and the Internet of Things used for control and automation. The progressive use of ICT technology exposed CPS to vulnerabilities and threats typical of the ICT world [1]–[3]. Cyber Physical Systems present many specific differences from standard ICT systems [4] that make general ICT security solutions seldom effective for CPS. However, such peculiarities can also lead to better tailored solutions.

CPS are typically designed for a specific purpose in a predetermined production environment. As a consequence, the behaviour of their physical process is predictable to a good extent and often well documented. Fortunately, this predictability reflects on the cyber counterpart, thus it is possible to leverage such knowledge of the CPS to specify known critical conditions that combine

David C. Wyld et al. (Eds) : CNSA, DMDB, AIAP - 2018
pp. 61– 77, 2018. © CS & IT-CSCP 2018 DOI : 10.5121/csit.2018.80105

cyber and process aspects for a greater expressiveness and effectiveness. However, to the author's 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.

It is necessary to observe the CPS current state to detect if it has reached a critical state. However, the assumption that every parameter of the CPS can always be observed is too strong and unfeasible in real cases. The main novelty of our approach is the ability to handle unobservable variables. Moreover, the framework is also capable of computing, if necessary, the piece of missing information required for a more accurate result. Such information is provided to security operators as a guide for finding a refinement of the CPS state. The refinement can feed the monitor back, leading to more precise detection.

Present paper improves our previous works [5] and present the same ability to predict whether the CPS is evolving towards some critical states, monitoring the changes in time of a notion of proximity from critical conditions. However, unobservable variables complicate the formal definition and the actual computation of the proximity, as explained in the following sections.
As previous works, the framework does not need a full model of the CPS, which is very hard to achieve in real cases. It is based on passive observations of the CPS through the analysis of network traffic and logs, to be more suitable for the industrial sector where change management and shutdowns are nearly impossible in practice, especially when employed in critical infrastructures. The framework presents an expressive specification language and is agnostic to observation methods and attack models, thus it is suitable for detecting possible 0-days attacks.

Section 2 shows a very simple example to explain the main idea behind the cyber security monitoring framework. Section 3 describes related works and approaches in literature and on the market. Section 4 defines our proposed framework, while Section 5 presents our first working prototype and our first experimental results that validates the approach.

2. A MOTIVATING EXAMPLE

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 sections refer to this example to ease the explanation.

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

- a thermometer in each room that measures the temperature T_i
- an external thermometer for the outdoor temperature O
- a radiator R_i in each room that can be switched on/off
- 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 [2], [6], a very widely used control protocol with no security mechanisms for authentication/authorisation.

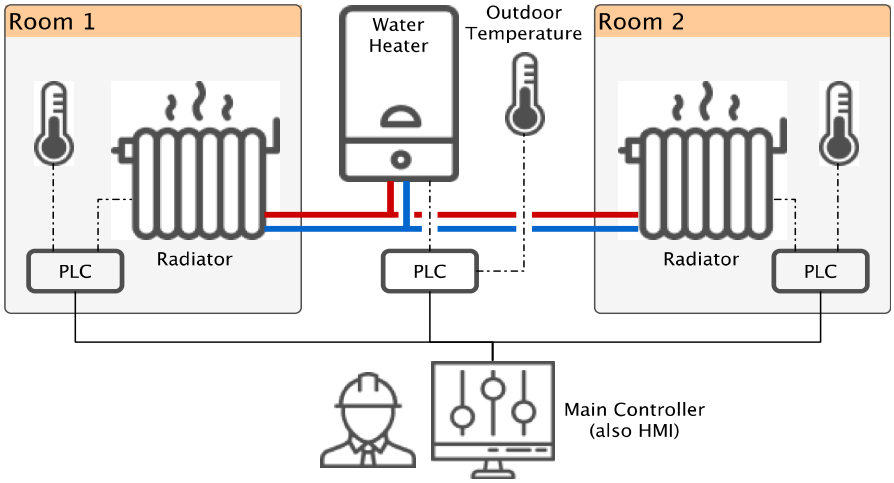

Figure 1. Two rooms building automation example.

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 operated through a set of rules implemented in the main controller MC and listed in Table 1.

Table 1. Example of automatic operation rules.

every 500 ms	→	read T_i, S_i, R_i, H, O
$T_i < S_i$ and $O < S_i$	→	write $R_i = \mathbf{true}$ and $H = \mathbf{true}$
$T_i > S_i$	→	write $R_i = \mathbf{false}$
$T_1 > S_1$ and $T_2 > S_2$	→	write $H = \mathbf{false}$

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 [7], [8], Suricata [9]) nor basic Modbus deep packet inspection tools (e.g. Wireshark [10]–[12], Bro [13]) 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, $f_{expected} = 3/0.5$ corresponds to 3 PLCs and 500 ms fromTable 1, and ϵ is a tolerance constant.

Similarly, suppose the attacker sends a Modbus write command $H = \mathbf{false}$ 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 1, expressed by the critical condition

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

Intuitively, critical condition(ϕ_2) holds if a turn off command is sent when the heater should be on according to Table 1.

Notice that the critical condition (ϕ_1) purely addresses the cyber layer of the CPS, while (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 (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

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

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 (ϕ_1) is not satisfied, a notion of *proximity* from (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 (ϕ_1).

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 [2], [3]. Literature presents several secured version of control protocol, e.g. [14]–[16]. 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 is 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 [7], [8], are able to express *bad* IP packet that can be detected. 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 [17]–[22]. [23] classifies anomaly-based IDS in two main categories:

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

ii. *specification-based techniques*, for which a human ICS 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 spot 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. [24]. 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 [25], [26], Elasticsearch-Logstash-Kibana (ELK) [27]–[30], Grafana [31], and LogRythm [32]. 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, security operators can 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 paper. Unlike our proposed framework, such tools allow queries only on observable data and do not offer a notion of proximity / proximity range from criticality.

Nai et al. [33]–[35] 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 [5] with a greater expressiveness and more effective computation methods. Our present work further improves the same approach. The novelty of this work consists in (i) dealing with unobservable aspects of the system for a greater expressiveness and feasibility in real cases (ii) using real-time knowledge refinements from human operators (iii) guiding the operator to express better refinements (iv) computing proximity and proximity ranges from criticality even in presence of unobservable variables.

4. THE MONITORING FRAMEWORK

The proposed monitoring framework passively runs in parallel with the monitored CPS. It continuously observes the current state of the CPS and checks it against conditions that are a-priori known to be illicit or anomalous, hereafter called *critical conditions*. Figure 2depicts the main structure of the framework.

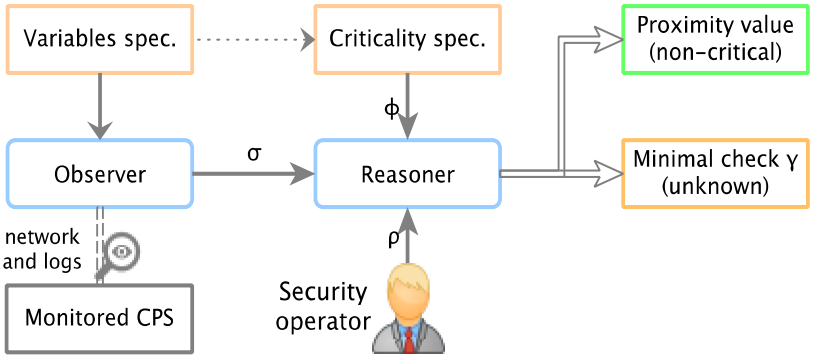

Figure 2. Structure of the real-time monitoring framework.

The first step is to identify the aspects of the CPS, called *variables*, that are necessary to express the critical conditions. In real cases, the assumption that it is always possible to retrieve the value of all the variables is too strict and unfeasible. Thus, a variable can be *observable* or *unobservable*, either temporarily or permanently. 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:

i. a variable bound to the value of a malfunctioning sensor that cannot provide its value;

ii. 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;

iii. 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 monitoring framework is composed by two main components: the *observer* and the *reasoner*. The former continuously analyses network traffic and logs generated by the monitored CPS, in order to retrieve the value of the observable variables. The latter checks the current state of the CPS against the set of known critical conditions.

The input to the observer consists of the *specification of variables*, which enumerates the variables of interest and their properties. Precisely it defines for each variable:

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

ii. the *type*: boolean, integer, or real

iii. an optional *range constraint*, i.e. lower and upper bounds

iv. an optional *observation method*: how the observer captures the value of the variable through network or log analysis. When the method fails or is not provided, the variable is considered unobservable.

This paper is agnostic w.r.t. observation methods, so any method can be used. The only constraint is that the observation needs to be nearly real-time and the result must be a boolean or numeric value. For instance, in our prototype the observer uses deep packet inspection against network traffic captured in real-time.

Our threat model assumes the integrity of the observed values: if an attacker takes the complete control of the network it might compromise the effectiveness and correctness of our monitoring framework. However, this assumption is typical of security monitoring solutions cited in Section 3. In real cases, such approach is still valid provided that a sufficient large number of variables are observable and effective critical conditions are specified. In this way the likelihood that an attacker is able to compromise enough values to make the monitor ineffective is low.

Iteratively the reasoner gets as input the values from the observer (σ) and a set of critical conditions (ϕ), and checks if each condition is met with the current observations. 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 human operators to give the monitor additional information about unobservable variable.

Each critical condition is associated to a function that represents a notion of *proximity*. When the reasoner is able to determine that CPS is currently not in a critical condition, it computes the proximity from that condition.

When the reasoner is unable to determine whether the current state satisfies a critical condition, it computes the minimal condition of unobservable variables that is necessary to determine that the system state is not critical (γ inFigure 2). The minimal condition γ is hereafter called *assisted*

check, because it helps security operators figure out the missing unobservable information. In other words, the assisted check can guide operators to provide better knowledge refinements.

4.1 Specification of Variables and Critical Conditions

Let $\mathcal{V}$ denote the set of variables, whose type can be boolean, integer, or real, and let $range(v)$ denote the range constraint of v defined in the variable specification. 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. As a consequence, all variables in $\mathcal{V}$ range on $\mathbb{R}$.

Definition 1. Let $V \subseteq \mathcal{V}$ be a subset of variables. A *partial assignment* (or simply an assignment) is a function $a: V \to \mathbb{R}$ that maps variables to their value such that $a(v) \in range(v)$ for each variable $v \in V$. The notation $dom(a)$ denotes its domain V.

Definition 2. A *state* of the monitored CPS (or simply a state) is an assignment s such that $dom(s) = \mathcal{V}$, i.e. a total assignment on variables. The set of all possible system states is denoted by $\mathcal{S}$. Given a partial assignment a, we define

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

as the set of states that satisfy the assignment a.

Our framework uses a partial assignment c to represent the current observations that the observer passes to the reasoner. If all variable are observable, c is total, i.e. c is a CPS state. In case of unobservable variables, c represents only the observable portion of the current state of the CPS, and from the perspective of the reasoner the CPS could be in any state of $\mathcal{S}(c)$.

Definition 3. A *state formula* is 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 $var(\phi)$.

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

Definition 4. Given a partial assignment c and a formula ϕ such that $var(\phi) \subseteq dom(c)$, the assignment c*satisfies* (or *models*) the formula ϕ, denoted by $c \vDash \phi$, when recursively:

$$c \vDash \sum_i a_i\, v_i \bowtie b \quad \text{iff} \quad \sum_i a_i\, c(v_i) \bowtie b \qquad\qquad c \vDash \neg\phi \quad \text{iff} \quad c \nvDash \phi$$

$$c \vDash \phi_1 \wedge \phi_2 \quad \text{iff} \quad c \vDash \phi_1 \text{ and } c \vDash \phi_2 \qquad\qquad c \vDash \phi_1 \vee \phi_2 \quad \text{iff} \quad c \vDash \phi_1 \text{ or } c \vDash \phi_2$$

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

Our framework uses state formulae to define the known critical conditions of the monitored CPS. Moreover, at each iteration the observer passes the receiver the formula

$$\sigma ::= \bigwedge_{v \in V} v = c(v)$$

representing the current observation, where V is the set of observed variables and $c(v)$ is their observed value.

Example. The simple example described in Section 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*

- *H: boolean variable corresponding to the on/off status of the heater*
- *M: boolean unobservable variable corresponding to the fact that the operator has manually and intentionally operated the CPS through the HMI.*

Variables T_i, S_i, O, H are observed through the analysis of the Modbus traffic on the process network.

4.2 Criticality Detection

Error! Reference source not found.depicts the behaviour of the reasoner at each iteration. For each critical formula ϕ, the reasoner uses the formula σ from the observer and an optional refinement ρ of the CPS state (if provided) to discriminate whether the monitored CPS has reached ϕ. To this aim, a formula is defined as

$$\kappa := \sigma \wedge \rho$$

that represents all the information about the CPS state s available to the reasoner, i.e. that $s \in \mathcal{S}(\kappa)$.

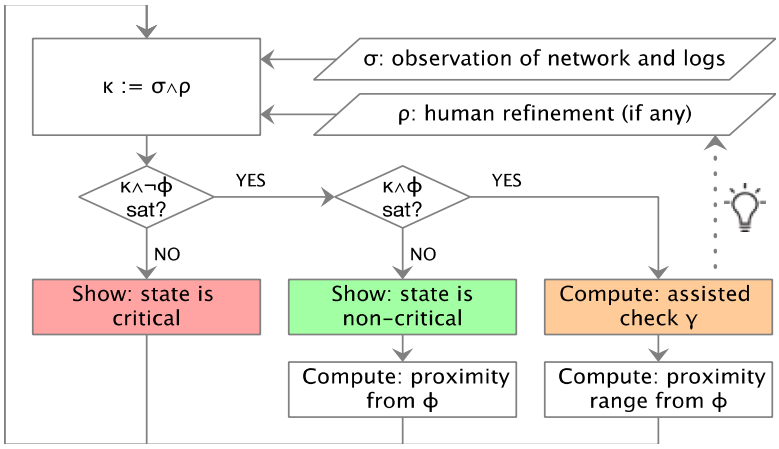

Figure 3. Reasoner flow chart given criticalityϕ.

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:

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

$$\kappa \wedge \neg\phi \text{ is unsatisfiable.} \quad (1)$$

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

$$\kappa \wedge \phi \text{ is unsatisfiable.} \quad (2)$$

iii. If both formulae in (1) and (2) are satisfiable, then $\mathcal{S}(\kappa) \cap \mathcal{S}(\phi) \neq \emptyset$ and $\mathcal{S}(\kappa) \backslash \mathcal{S}(\phi) \neq \emptyset$. 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.

The last case means that κ does not contain enough information to discriminate the criticality of the CPS state. Since the observation σ does not contain information about unobservables by definition, the only way to obtain a more precise result is to provide a more informative refinementρ.

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 minimal condition, hereafter denoted by γ, that is sufficient to guarantee the non criticality of the current CPS state given κ and ϕ, i.e. such that $\kappa \wedge \gamma \wedge \phi$ is not satisfiable. Our monitoring solution shows γ to a human operator, who can try to manually check the CPS in order to verify if γ holds, or at least if some of the sub-formulae ofγ hold. This way the operator may acquire some information and 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, thus it is crucial that γ is minimal.

We use the notion of interpolant, provided by most SMT solvers, to compute the minimal assisted check γ. Given two mutually unsatisfiable formulae α and β, a *Craig interpolant* (denoted by $interpolant(\alpha, \beta)$) is a formula η such that $var(\eta) \subseteq var(\alpha) \cap var(\beta)$ and formulae $\alpha \rightarrow \eta$ and $\eta \rightarrow \neg\beta$ 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.

Our framework also uses syntactic simplification of logical expressions that most SMT solvers provide. Hereafter $simplify(\alpha)$ denotes the computation of a possibly simpler expression equivalent to α. The analysis of the most effective simplification tactics goes beyond the aim of this work[1] and does not compromise the soundness of our approach.

Since formulae $\kappa \wedge \neg\phi$ and $\kappa \wedge \phi$ are mutually unsatisfiable, the assisted check can be defined as

$$\gamma \coloneqq interpolant(simplify(\kappa \wedge \neg\phi), simplify(\kappa \wedge \phi)) \tag{3}$$

Example. Assume that the example CPS of Section 2 reaches a state where the room temperatures are $15\,^{\circ}C$ *and* $23\,^{\circ}C$*, the desired temperatures are* $21\,^{\circ}C$ *and* $17\,^{\circ}C$*, 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*

$$\sigma \coloneqq \neg H \wedge T_1 = 15 \wedge S_1 = 21 \wedge T_2 = 23 \wedge S_2 = 17 \tag{4}$$

Assume no further refinement is provided, i.e. $\kappa = \sigma$*. 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 to tell if the CPS is in a critical state, because this depends on the actual value of the unobservable variables* M *and* O*. In this case, the framework computes the assisted check*

$$\gamma \coloneqq M \vee O \geq 21$$

Indeed, in order 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 thanS_1*, which is 21. Notice that* γ *is minimal, i.e. it only contains the unobservable variables. If the operator manually operated the system, he/she can provide the refinement* $\rho \coloneqq M$*.*

4.3 Predictiveness: Proximity from Critical Conditions

In this section we define the notion of proximity from a critical condition ϕ. Given a set X, a function $d: X \times X \rightarrow \mathbb{R}$ 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 $d: X \times X \rightarrow \mathbb{R}$ is called a *metric* if for all $x, y, z \in X$: (i)

[1] As a reference, our prototype uses Z3 [36] with the tactic `(then simplify ctx-simplify ctx-solver-simplify)`.

$d(x,y) = 0$ iff $x = y$, (ii) $d(x,y) = d(y,x)$, (iii) $d(x,y) \leq d(x,z) + d(z,y)$. The pair (X,d) is called *metric space*.

We use the following well known result. Let (X,d) be a metric space. The function $D: 2^X \times 2^X \to \mathbb{R}$ defined as

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

is a premetric.

Provided any enumeration of the CPS variables $\mathcal{V}$, the set of states $\mathcal{S}$ can also be seen as a vector of $\mathbb{R}^n$, where n is the number of variables. Thus, any metric d on $\mathbb{R}^n$ is a metric on $\mathcal{S}$ that induces a premetric D on $2^{\mathcal{S}}$. In the following we use the premetric D to capture the notion of proximity from critical condition.

Our framework requires to specify for each critical condition ϕ an associated metric d. Recall that at runtime the formula $\kappa \coloneqq \sigma \wedge \rho$ represents what the reasoner knowns about CPS variables. The *proximity* of the current CPS state from the critical condition ϕ is defined as

$$D(\mathcal{S}(\kappa), \mathcal{S}(\phi))$$

hereafter denoted by $D(\kappa, \phi)$.

Previous definition is parametric w.r.t. the chosen metric on the set of states $\mathcal{S}$, and the actual choice function depends on the application. Table 2shows possible examples of metrics. For instances, the Hamming distance captures the number of variables that differs, while the Manhattan distance captures each variable variation, and this choice allows for a qualitative vs. quantitative proximity notion.

Table 2. Example of metrics on $\mathcal{S}$.

$d(s,t) = \sum_{v \in V} \lvert s(v) - t(v) \rvert$	Manhattan distance (i. e. L_1 metric on $\mathbb{R}^n$)
$m_V(s,t) = \frac{1}{\#V} \sum_{v \in V} \frac{\lvert s(v) - t(v) \rvert}{v_{\max} - v_{\min}}$	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
$nh_V(s,t) = \frac{1}{\#V} \cdot \#\{v \in V \mid s(v) \neq t(v)\}$	Normalised Hamming distance

where $s, t \in \mathcal{S}$, $V \subseteq \mathcal{V}$, $v_{\min} = \min(range(v))$, $v_{\min} = \max(range(v))$.

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 [36] and by OptiMathSat [37]. Figure 4shows the pseudo-algorithm to compute the proximity $D(\kappa, \phi)$ given the current knowledge of the system κ and the criticality expressed by the formula ϕ.

```
function Proximity(κ, ϕ)
   (S, T) ← two distinct sets of fresh symbols for variables
   distance ← new real symbol
   solver ← new SMT-Optimizing-Solver
   solver.assert κ[𝒱 ↦ S] ∧ ϕ[𝒱 ↦ T]
   solver.assert distance = metric(S, T)
   solver.goal ← minimize(distance)
   model ← solver.check-sat()
if model not foundthen
return Error: either κ or ϕ is unsatisfiable
else
returnmodel.getvalue(distance)
```

Figure 4. Proximity pseudo-algorithm.

When the reasoner is not able to discriminate that the CPS is not in a critical condition, i.e. when both $\kappa \wedge \phi$ and $\kappa \wedge \neg\phi$ are satisfiable, it is necessary to compute a notion of *proximity range* from the critical condition ϕ that handles the missing information about unobservable variables, defined as the pair $(D_{\min}, D_{\max})$ where

$$D_{\min} = \inf_{s \in \mathcal{S}(\kappa) \setminus \mathcal{S}(\phi)} D(\{s\}, \phi) = \inf_{\substack{s \vDash \kappa \wedge \neg\phi \\ t \vDash \phi}} d(s,t)$$

$$D_{\max} = \sup_{s \in \mathcal{S}(\kappa) \setminus \mathcal{S}(\phi)} D(\{s\}, \phi) = \sup_{s \vDash \kappa \wedge \neg\phi} \inf_{t \vDash \phi} d(s,t)$$

Computing $D_{\min}$ corresponds to the same optimisation problem on linear constraints as before, while computing $D_{\max}$ requires to iteratively search the maximum results of the same optimization problem, increasing the distance until no result if found. Figure 5shows the pseudo-algorithm to compute the proximity range $(D_{\min}, D_{\max})$.

```
functionProximityRange(κ, ϕ)
   (S, T) ← two distinct sets of fresh symbols for variables
   distance ← new real symbol
   solver ← new SMT-Optimizing-Solver
   solver.assert κ[𝒱 ↦ S] ∧ ¬ϕ[𝒱 ↦ S]
   solver.assert ϕ[𝒱 ↦ T]
   solver.assert distance = metric(S, T)
   solver.goal ← minimize(distance)
   model ← solver.check-sat()
if model not found then
return Error: either κ ∧ ¬ϕ or ϕ are unsatisfiable
else
mindistance← model.getvalue(distance)
repeat
maxdistance← model.getvalue(distance)
forv ∈ 𝒱do
solver.assert T(v) ≠ model.getvalue(T(v))
solver.assertdistance > maxdistance
model ← solver.check-sat()
until model is not found
return (mindistance, maxdistance)
```

Figure 5. Proximity range pseudo-algorithm.

Example. Assume κ is defined as (4) in the previous example. Recall that both $\kappa \wedge \phi_3$ and $\kappa \wedge \neg\phi_3$ are satisfiable. Using the Hamming distance $h_{\mathcal{V}}$ on the seven variables T_i, S_i, O, H, and M, the value of the proximity and the proximity range from ϕ_3 are

$$D(\kappa, \phi_3) \quad = 0 \quad D_{min} \quad = \frac{1}{7} \quad D_{max} \quad = \frac{2}{7}$$

The fact that the proximity $D(\kappa, \phi_3) = 0$ is correct, since the current CPS state could be critical depending on the actual value of the unobservable variables M and O. For this reason, the reasoner computes the proximity range instead of the proximity. Proximity range gives pessimistic and optimistic estimation of the proximity, under the assumption that the current state is not critical: values 1/7 and 2/7 correctly indicates that the number of variables (out of seven) required to change before reaching ϕ_3 is 1 or 2 respectively.

5. EXPERIMENTAL RESULTS

This section briefly shows how our first prototype 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 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 [39]. Physics is simulated using the Newton's Law of Cooling, often used in literature (e.g. [40]).

- **nodered**: the HMI, the human manual operations, and the automatic operation logic of Table 1, and the attackers command are implemented using *Node-RED* [41], 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 [36].

- **influxdb** [42] and **grafana** [31]: a time series database and a data visualisation software that provide the graphical user interface. Figure 6 shows a screenshot.

In this way the environment is able to simulate the main components depicted in Figure 1and 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 sends the value to the reasoner via the local MQTT server on the `sensor/T1` topic.

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 [10], to extract the values of interest through its basic deep packet inspection functionalities. The observer and the reasoner are two distinct pieces of software that communicate through the MQTT sub/pub protocol [43] (QoS 1).

The reasoner component is a Python application that implements Figure 3. 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 [36] with the Python API to evaluates the

criticality of the CPS state, to compute the minimal assisted check defined in (3), and to compute proximity and proximity range from the critical condition as Figure 4 and Figure 5.

Figure 6. Screenshot of the graphical interface of the monitoring prototype.

Figure 6 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 Section 4.2 and in the flow chart in Figure 3. The forth column is empty in case of "critical" and "non critical", and contains the minimal assisted check γ described in Section 4.2. The fifth column contains the user refinement ρ, if provided.

The rest of the graphical dashboard shows the proximity range[2]. The two gauges represent the real-time values of $D_{\min}$ and $D_{\max}$. The chart shows the timeseries of $D_{\min}$ and $D_{\max}$. It is easy to see that the values are constantly decreasing, thus the system is getting closer to the critical condition. In the leftmost part of the chart $D_{\min}$ and $D_{\max}$ are both close to 1, hence the current state of the monitored CPS is distant from the critical states and unobservable variables have a low impact. The central part of the chart shows that $D_{\min}$ and $D_{\max}$ greatly differ each other: this means that unobservable variables have a bigger role on the actual proximity of the CPS from the critical condition. 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 close to the criticality, because both $D_{\min}$ and $D_{\max}$ 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.

[2] Values shown here are only for example purposes and for a better graphical explanation.

6. FINAL REMARKS

This work presents a specification-based predictive cyber security monitoring framework for cyber physical systems and improves [5]. It enables specifying known critical conditions, through an easy but expressive formal language, that 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 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 the minimal piece of information that is required to discriminate the criticality of the CPS state, and provide such information as a guide to the operator.

Unobservable variables also complicate computing the proximity from critical states. However, the framework is able to compute a min/max range of the distance from critical states. Our working prototype also presents a graphical interface showing the history of the proximity range, providing an overview of the evolution of the system w.r.t. the specified critical conditions.

This work uses SMT techniques to assess the criticality of the CPS current state and to compute the minimal assisted checks. It also uses SMT-based optimisation techniques to compute proximity ranges from critical states. Preliminary results proves an expressive specification language and an efficient reasoning engine. While first results seem feasible an promising, precise experiments will be the subject of further investigation to assess the limits of our approach.

Another aspect to further improve is the characterization of cyber metrics and aggregate functions that can be leverage in the monitoring framework. In particularly, security monitoring common in the DevOps sector provide a plethora of metrics and functions that can be integrated in our framework. The framework only permits to specify linear constraints and distances computable through linear optimisation problems. Another subject to further investigate are recent works about satisfiability modulo linear and non-linear theories over reals [44]–[48].

REFERENCES

[1] V. M. Igure, S. A. Laughter, and R. D. Williams, “Security issues in SCADA networks,” *Computers & Security*, vol. 25, no. 7, pp. 498–506, Oct. 2006.

[2] P. Huitsing, R. Chandia, M. Papa, and S. Shenoi, “Attack taxonomies for the Modbus protocols,” *International Journal of Critical Infrastructure Protection*, vol. 1, pp. 37–44, Dec. 2008.

[3] S. East, J. Butts, M. Papa, and S. Shenoi, “A Taxonomy of Attacks on the DNP3 Protocol,” in *Critical infrastructure protection iii*, 2009, pp. 67–81.

[4] K. Stouffer, V. Pillitteri, S. Lightman, M. Abrams, and A. Hahn, “Guide to Industrial Control Systems (ICS) Security,” National Institute of Standards; Technology, Gaithersburg, MD, Jun. 2015.

[5] A. Coletta and A. Armando, "Security Monitoring for Industrial Control Systems," in *Security of industrial control systems and cyber physical systems. CyberICS 2015*, 2016, pp. 48–62.

[6] "MODBUS Application Protocol Specification V1.1b3," 2012.

[7] M. Roesch, "Snort: Lightweight Intrusion Detection for Networks." *LISA '99: 13th Systems Administration Conference*, pp. 229–238, 1999.

[8] B. Caswell and J. Beale, *Snort 2.1 intrusion detection*. Syngress, 2004.

[9] Suricata, "Suricata Open Source IDS / IPS / NSM engine." 2017.

[10] G. Combs and Others, "Wireshark," *www.wireshark.org*, 2017.

[11] A. Orebaugh, G. Ramirez, and J. Beale, *Wireshark & Ethereal network protocol analyzer toolkit*. Syngress, 2006.

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

[13] V. Paxson, "Bro: a system for detecting network intruders in real-time," *Computer Networks*, vol. 31, nos. 23-24, pp. 2435–2463, 1999.

[14] I N. Fovino, A. Carcano, M. Masera, and A. Trombetta, "Design and Implementation of a Secure Modbus Protocol," in *International conference on critical infrastructure protection*, 2009, pp. 83–96.

[15] M. Majdalawieh, F. Parisi-Presicce, and D. Wijesekera, "DNPSec: Distributed network protocol version 3 (DNP3) security framework," in *Advances in computer, information, and systems sciences, and engineering*, Springer, 2007, pp. 227–234.

[16] G. Gilchrist, "Secure authentication for DNP3," in IEEE power and energy society general meeting - conversion and delivery of electrical energy in the 21st century, 2008, pp. 1–3.

[17] D. Bolzoni, S. Etalle, P. Hartel, and E. Zambon, "POSEIDON: a 2-tier Anomaly-based Network Intrusion Detection System," in *Fourth ieee international workshop on information assurance (iwia)*, 2006.

[18] W. Heimerdinger, V. Guralnik, and R. VanRiper, "Anomaly-based intrusion detection." Google Patents, 2006.

[19] C. Zimmer, B. Bhat, F. Mueller, and S. Mohan, "Time-based intrusion detection in cyber-physical systems," *Proceedings of the 1st ACM/IEEE International Conference on Cyber-Physical Systems - ICCPS '10*, p. 109, 2010.

[20] S. Cheung, B. Dutertre, M. Fong, U. Lindqvist, K. Skinner, and A. Valdes, "Using Model-based Intrusion Detection for SCADA Networks," *Science And Technology*, vol. 329, pp. 1–12, 2006.

[21] R. Mitchell and I. R. Chen, "Behavior Rule Specification-Based Intrusion Detection for Safety Critical Medical Cyber Physical Systems," *IEEE Transactions on Dependable and Secure Computing*, vol. 12, no. 1, pp. 16–30, 2015.

[22] K. Xiao *et al.*, "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)*, 2007, pp. 4–4.

[23] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," *Computers & Security*, vol. 28, nos. 1-2, pp. 18–28, Feb. 2009.

[24] M. Caselli *et al.*, "Specification Mining for Intrusion Detection in Networked Control Systems Specification Mining for Intrusion Detection in Networked Control Systems," *Proceedings of the 25th USENIX Security Symposium*, pp. 791–806, 2016.

[25] D. Carasso, *Exploring Splunk*. CITO Research, 2012.

[26] J. Diakun, P. R. Johnson, and D. Mock, *Splunk Operational Intelligence Cookbook*. Packt Publishing Ltd, 2016.

[27] C. Gormley and Z. Tong, *Elasticsearch: the Definitive Guide*. O'Reilly Media, Inc., 2015.

[28] J. Turnbull, *The Logstash Book*. James Turnbull, 2013.

[29] Y. Gupta, *Kibana Essentials*. Packt Publishing Ltd, 2015.

[30] G. S. Sachdeva, *Practical ELK Stack*. Apress, 2017.

[31] Grafana Labs, "Grafana." 2017.

[32] LogRhythmInc, "LogRhythm security intelligence and analytics platform." 2017.

[33] I. NaiFovino, A. Coletta, A. Carcano, and M. Masera, "Critical State-Based Filtering System for Securing SCADA Network Protocols," *IEEE Transactions on Industrial Electronics*, vol. 59, no. 10, pp. 3943–3950, Oct. 2012.

[34] A. Carcano, A. Coletta, M. Guglielmi, M. Masera, I. NaiFovino, and A. Trombetta, "A Multidimensional Critical State Analysis for Detecting Intrusions in SCADA Systems," *IEEE Transactions on Industrial Informatics*, 2011.

[35] I. NaiFovino, A. Carcano, A. Coletta, M. Guglielmi, M. Masera, and A. Trombetta, "State-Based Firewall for Industrial Protocols with Critical-State Prediction Monitor," in *Critical information infrastructures security*, vol. 6712 LNCS, 2011, pp. 116–127.

[36] L. De Moura and N. Bjørner, "Z3: An efficient SMT solver," in International conference on tools and algorithms for the construction and analysis of systems, 2008, pp. 337–340.

[37] R. Sebastiani and P. Trentin, "OptiMathSAT: A Tool for Optimization Modulo Theories," in *International conference on computer aided verification*, 2015, pp. 447–454.

[38] Docker Inc, "Docker." 2017.

[39] G. Collins, "Pymodbus 1.2.0." 2017.

[40] A. Cole, B. Jury, and K. Takashina, "A Leidenfrost Thermostat," *Journal of Heat Transfer*, vol. 137, no. 3, p. 034502, Mar. 2015.

[41] JS Foundation, "Node-RED." 2017.

[42] InfluxDataInc, "InfluxDB." 2017.

[43] A. Banks and R. Gupta, "MQTT Version 3.1. 1," *OASIS standard*, 2014.

[44] S. Gao, S. Kong, and E. Clarke, "Satisfiability Modulo ODEs," in *Formal methods in computer-aided design*, 2013, pp. 105–112.

[45] S. Kong, S. Gao, W. Chen, and E. Clarke, "dReach: δ-Reachability Analysis for Hybrid Systems," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2015, pp. 200–205.

[46] S. Gao, L. Xie, A. Solar-Lezama, D. Serpanos, and H. Shrobe, "Automated vulnerability analysis of AC state estimation under constrained false data injection in electric power systems," in *2015 54th ieee conference on decision and control (cdc)*, 2015, vols. 2016-Febru, pp. 2613–2620.

[47] K. Bae, P. C. Ölveczky, S. Kong, S. Gao, and E. M. Clarke, "SMT-Based Analysis of Virtually Synchronous Distributed Hybrid Systems," in *Proceedings of the 19th international conference on hybrid systems: Computation and control - hscc '16*, 2016, pp. 145–154.

[48] S. Gao and D. Zufferey, "Interpolants in Nonlinear Theories Over the Reals," in Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics), vol. 9636, 2016, pp. 625–641.

AUTHOR

Alessio Coletta possesses a Master Degree in Computer Science at the ScuolaNormaleSuperiore di Pisa and a Master Degree in Information Security at the Royal Holloway University of London. He has worked at the Joint Research Centre of the European Commission as a scientific officer in the Security of Networked Critical Infrastructure unit. He has also worked in the Global Cyber Security Center foundation of Poste Italiane and in the Incident Prevention and Management structure of Poste Italiane. He is a PhD candidate at the University of Trento performing R&D activities on security of cyber physical system within the Security and Trust unit of the Fondazione Bruno Kessler in Trento (Italy).