PREDICTING SECURITY CRITICAL CONDITIONS OF CYBER PHYSICAL SYSTEMS WITH UNOBSERVABLES AND OBSERVATION TIMES

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

Cyber Physical Systems (CPS), like IoT and industrial control systems, are typically vulnerable to cyber threats due to a lack of cyber security measures and hard change management. Security monitoring is aimed at improving the situational awareness and the resilience to cyber attacks. Solutions tailored to CPS are required for greater effectiveness. This work proposes a monitoring framework that leverages the knowledge of the system to monitor in order to specify, check, and predict known critical conditions. This approach is particularly suitable to CPS, as they are designed for a precise purpose, well documented, and predictable to a good extent. The framework uses a formal logical language to specify quantitative critical conditions and an optimisation SMT-based engine that checks observable aspects from network traffic and logs. The framework computes a quantitative measure of the criticality of the current CPS system: checking how criticality changes in time enables to predict whether the system is approaching to a critical condition or reaching back a licit state. An important novelty of the approach is the capability of expressing conditions on the time of the observations and of dealing with unobservable variables. This work presents the formal framework, a prototype, a testbed, and first experimental results that validate the feasibility of the approach.

KEYWORDS

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

1. INTRODUCTION

Cyber Physical Systems (CPS) include Industrial Control Systems (ICS), Internet of Things, and critical infrastructures. They are composed by networked ICT devices that support the operation of physical entities and are employed in a large number of business- or safety-critical sectors. Due to their historical evolution, the progressive use of ICT technology without proper cyber security measures exposed CPS to vulnerabilities and threats typical of the ICT world [1-3]. CPS present many specific differences from standard ICT systems [4] that make general ICT security solutions seldom effective for CPS. Fortunately, the same peculiarities can also lead to better tailored solutions.

CPS are aimed at a specific purpose in a determined environment. As a consequence, the behaviour of their physical process is well designed and predictable to a good extent, and typically well documented. Also the behaviour of the cyber counterpart is predictable: a security operator may use such knowledge to specify critical conditions to be monitored. It is also possible to combine cyber and process aspects for a greater expressiveness and effectiveness.

While a number of the statistical and anomaly detection solutions are present in the literature and in the market, specification-based security monitoring approaches appear less mature. This work contributes in this regard presenting a framework that enables a security operator specifying which aspects of the CPS to observed, to express logical and quantitative critical condition about observed variables, and to detect and predict the criticality of the current state of the CPS.

The assumption that every parameter of the CPS can always be observed is not suitable to real cases. The main novelty of our approach is the ability to handle unobservable variables. Present work improves our previous results in that regard, defining a quantitative criticality notion whose changes in time predict both whether the CPS is getting closer to a critical condition and whether it is returning back to licit states. In presence of unobservable variables, the framework is capable of computing the criticality in the best and worst cases. It also computes the piece of missing information required for a more accurate result as a logical expression of unobservable values. Such information is provided to security operators as a guide for finding a refinement of the CPS state.

Present paper improves our previous works [5] and [6] in these aspects. Moreover, in this work the reasoning is untangled from observations, and it is possible to specify critical condition that depends on properties of observation times. This enables detecting illicit behaviours that depends on their time evolution properties.

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 describes related existing works and approaches. Section 3 shows an example to explain the main idea behind the cyber security monitoring framework. The same example is also used as a simulation scenario for our feasibility and performance testbed. Section 4 defines our proposed framework, while Section 5 presents our first working prototype and our first experimental results that validates the approach.

2. 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. [7-9]. 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 [10], [11], 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 [12-17]. [18] 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 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. [19]. 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 include Splunk [20], [21], Elasticsearch-Logstash-Kibana (ELK) [22-25], Grafana [26], and LogRythm [27]. 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. [28-30] 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 be monitored, 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 (1) using observation times untangled from reasoning (2) dealing with unobservable aspects of the system for a greater expressiveness and feasibility in real cases (3) using real-time knowledge refinements from human operators (4) guiding the operator towards better refinements (5) computing and monitoring a the criticality of the CPS state in the best and worst case to predict whether it is getting closer to critical conditions or returning back to licit states even in presence of unobservable variables.

3. A MOTIVATING EXAMPLE

This section presents an example of a simplified chemical process and its control system and logic. This scenario serves both to explain our proposed approach and as simulation scenario to test our prototype and validate the results. Figure 1 shows the components of the chemical process.

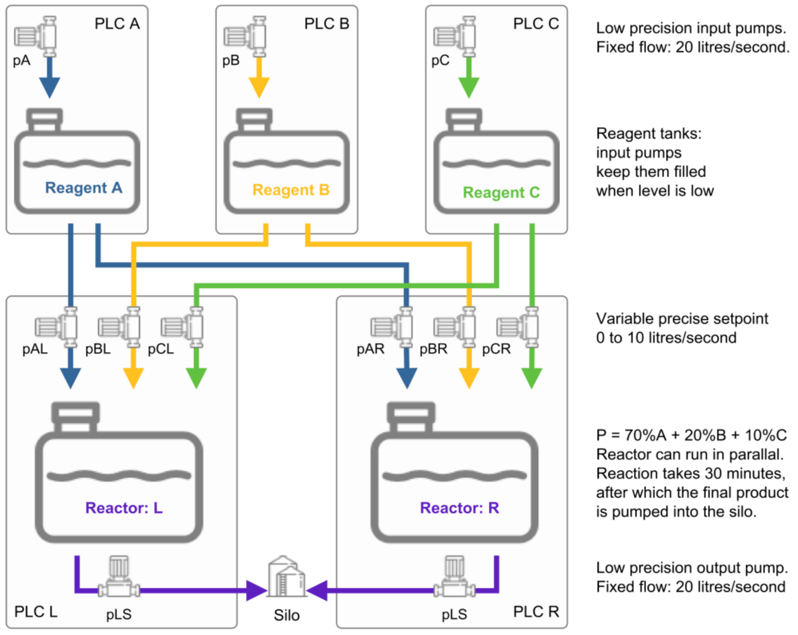
PLC A
pA
Reagent A
PLC B
pB
Reagent B
PLC C
pC
Reagent C
Low precision input pumps. Fixed flow: 20 litres/second.
Reagent tanks: input pumps keep them filled when level is low
pAL
pBL
pCL
pAR
pBR
pCR
Variable precise setpoint 0 to 10 litres/second
Reactor: L
Reactor: R
P = 70%A + 20%B + 10%C Reactor can run in parallal. Reaction takes 30 minutes, after which the final product is pumped into the silo.
PLC L
pLS
Silo
pLS
PLC R
Low precision output pump. Fixed flow: 20 litres/second

Figure 1. Simple chemical process use case.

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 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 [31]:

- 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 [32]. 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

$$RF_A \neq rf_A \vee RF_B \neq rf_B \vee RF_C \neq rf_C \vee RF_L \neq rf_L \vee RF_R \neq rf_R \tag{1}$$

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:

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

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 2 shows the state-based rules for sensors and actuators that control the reaction, while Table 1 shows the rules to control the other components.

Table 1. Example of automatic operation rules.

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

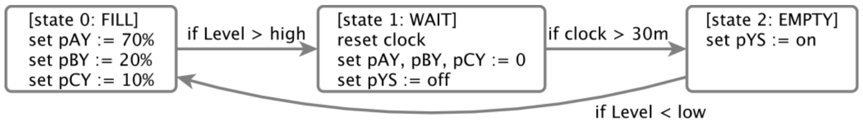

Figure 2. Automatic reactor control statechart ($Y \in \{L, R\}$).

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:

$$((S_L = 0 \lor S_L = 1) \land pLS = \text{on}) \lor (S_L = 2 \land pLS = \text{off}) \tag{3}$$

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 (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 2, 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:

$$\neg(pLSon.t < pLSoff.t \quad \rightarrow \quad 30m < pLSoff.t - pLSon.t < 31m) \tag{4}$$

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 $S_L = 0 \lor S_L = 2$.

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 (2) (3) (4) do not discriminate such licit commands from the attacker's ones. Each condition ϕ can be replaced with $\neg M \rightarrow \phi$, 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 (1), its criticality measure represents how the observed polling differs from the expected one. On the other hand, if the current state is not critical w.r.t. (4), i.e. the observation time between on and off commands is between 30 and 31 minutes, the proximity from the criticality represent how the observation time difference is close to the critical boundaries 30 or 31.

Unobservable variables imply that the current state is not fully known, hence the criticality or the proximity can be evaluated on a range of possible values. The following sections shows how our monitor computes the criticality/proximity range in the best and worst case.

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 the specified *critical conditions*. Figure 3 depicts the main structure of the framework.

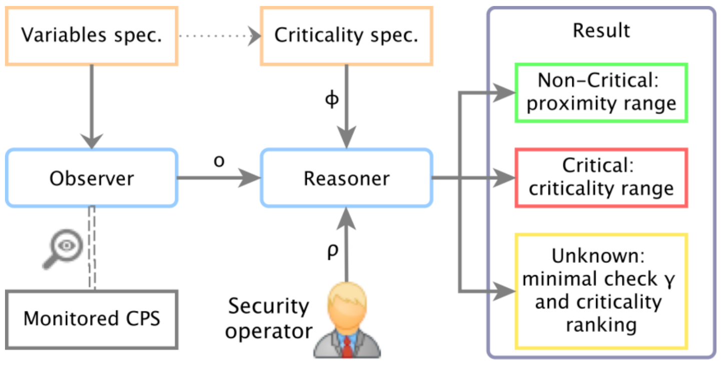
Variables spec.
Criticality spec.
Result
ϕ
Non-Critical: proximity range
o
Observer
Reasoner
Critical: criticality range
ρ
Unknown: minimal check γ and criticality ranking
Monitored CPS
Security operator

Figure 3. 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:

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 monitoring framework is composed by two main components: the *observer* and the *reasoner*.

The former continuously observes the CPS, e.g. analysing the traffic on the control network, 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:

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

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

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

4. 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.

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 2. 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 receives the observation o and 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 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 the minimal condition of unobservable variables that is necessary to determine that the system state is not critical (γ in Figure 3). 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 $\mathbb{R}$.

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 $o: V \to \mathbb{R}$ and $o^t: V \to \mathbb{T}$ such that $o(v) \in range(v)$ for each variable $v \in V$. The notation $dom(o)$ denotes its domain V.
When clear from the context we use o to indicate the pair (o, o^t).

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 $dom(s) = \mathcal{V}$. Given an observation o, we define

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

as the set of states that coincide with s.

The reasoner regularly receives the most current observation o from the observer. If $v \notin dom(o)$ 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.

A *critical condition formula* is defined by the grammar:

$$\begin{array}{lll} X & ::= v \mid v.t \mid \text{now} & \text{value or timestamp of variable observation} \\ \phi & ::= a_1X_1 + \cdots + a_nX_n \bowtie b \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi & \end{array} \tag{5}$$

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

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 current 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.

Definition 1. Given an observation o, a point in time τ, and a formula ϕ such that $var(\phi) \subseteq dom(o)$, the observation o *satisfies* (or *models*) at time τ the formula ϕ, denoted by $o, \tau \vDash \phi$, when recursively:

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

$$\begin{array}{rll} o,\tau \vDash \sum_i a_i X_i \bowtie b & \text{iff} & \sum_i a_i [\![X_i]\!]_{o,\tau} \bowtie b \\ o,\tau \vDash \neg\phi & \text{iff} & o,\tau \nvDash \phi \\ o,\tau \vDash \phi_1 \wedge \phi_2 & \text{iff} & o,\tau \vDash \phi_1 \text{ and } o,\tau \vDash \phi_2 \\ o,\tau \vDash \phi_1 \vee \phi_2 & \text{iff} & o,\tau \vDash \phi_1 \text{ or } o,\tau \vDash \phi_2 \end{array}$$

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. The reasoner iteratively receives from the observer the most recent observation o, and try to evaluate if $o, \tau \vDash \phi$ 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 $o, \tau \vDash \phi$ due to unobservable variables. Formally, if $var(\phi) \subseteq dom(o)$ then $o, \tau \vDash \phi$ can be simply checked using semantics in **Definition 1** defined by induction on the syntax of the formula. Otherwise, it may not be possible to check its satisfiability. The following section describes how the reasoner handles the satisfiability of critical conditions in order to detect when critical condition occurs and to measure the criticality of the current state of the CPS (or its distance from the critical condition).

4.2. The Observer

The proposed monitoring framework is agnostic to actual observation methods. This section describes assumptions and give examples of possible methods feasible to cyber physical systems. The critical specification language defined in (5) is able to express the observed value and the observation time of variables. An observation point is timestamped value (t, x), with the meaning that value x was actually observed at that time t. Variable specifications associate each variable v with an observation method, i.e. any procedures that returns the most recent observation point. The method may fail to represent unobservable variables. In this case the Observer does not pass any observation point for that variable to the Reasoner.

The reasoning time, i.e. the time the Reasoner computes the criticality of the current state, generally does not coincide with observation time. This is typical in CPS, where the supervisor server polls PLCs to collect the process values independently from any possible analysis. Moreover, this enables passive observations using application logs and/or network traffic analysis, as in our testbed described in Section 5. For this reason, the observer and the reasoner must be untangled. Established continuous monitoring solutions, largely employed in the industry, maintain the data collection separated from the analysis using efficient storage in the middle. We use the same approach to keep the observer and the reasoner untangled, since it proved to be very effective and scalable. Observations are stored in timeseries databases, i.e. in databases that provide efficient methods to store and retrieve timestamped data with an ad-hoc query language.

In this work we assume that all the observations are stored in one or more timeseries databases that represent application logs and parsed network traffic dumps. In particular, the observations in the use case described in Section 3 are the parsed Modbus-like messages dumped from the Process Control Network of the form:

```
READ <SESSIONID> <PLC> <PARAMETER>
READ_RESPONSE <SESSIONID> <PLC> <PARAMETER> <RETURN VALUE> <RESPONSE STATUS>
WRITE <SESSIONID> <PLC> <PARAMETER> <NEW VALUE>
WRITE_RESPONSE <SESSIONID> <PLC> <PARAMETER> <RESPONSE STATUS>
```

Table 2 shows an example of data stored in the database that represents messages observed in the PCN.

Table 2. Example of observations of PCN traffic as parsed network messages stored in a timeseries DB.

Time	Msg Type	Session	PLC	Parameter	Value	Status
t1	WRITE	342	PLC_A	pump_onoff	1	null
t2	WRITE_RESPONSE	342	PLC_A	pump_onoff	null	OK
t3	READ	343	PLC_C	pump_onoff	null	null
t4	READ_RESPONSE	343	PLC_C	pump_onoff	1	OK
t5	READ	345	PLC_L	tank_level	null	null
t6	READ_RESPONSE	345	PLC_L	tank_level	null	ERROR

The observation method of variables can be implemented using a query to the timeseries database that returns at most one timestamped value. For instance, the observation method of variable $pLSon$ occurring in (4) can be implemented with the query

```
SELECT time, value FROM pcn_db
WHERE msg_type="WRITE" AND plc="PLC_L" AND parameter="pump" AND value="1"
LIMIT 1
```

which retrieves only the time and value attributes of the most recent entry (`LIMIT 1`) that refers to a write command to pLS with payload value is 1 (i.e. on) from the database containing PCN messages such that the PLC.

4.3. The Reasoner

Figure 4 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.

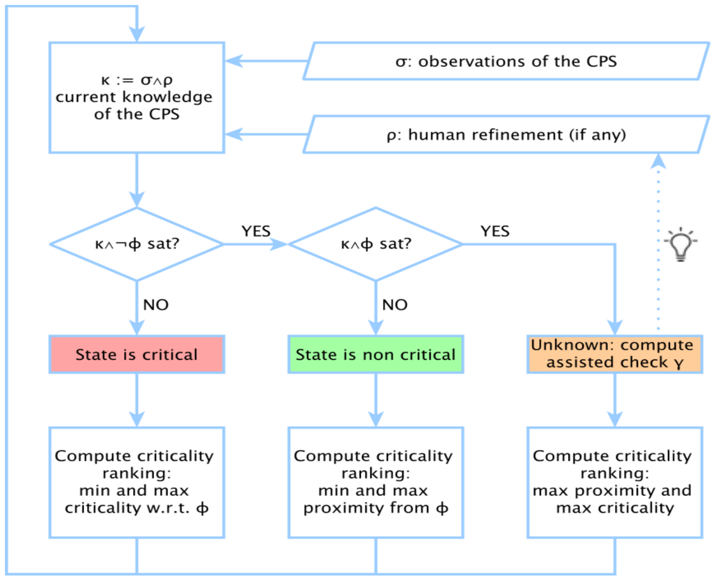
κ := σ∧ρ current knowledge of the CPS
σ: observations of the CPS
ρ: human refinement (if any)
κ∧¬ϕ sat?
YES
κ∧ϕ sat?
YES
NO
NO
State is critical
State is non critical
Unknown: compute assisted check γ
Compute criticality ranking: min and max criticality w.r.t. ϕ
Compute criticality ranking: min and max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

Figure 4. Reasoner flow chart given criticality ϕ.

The core of the reasoner is a Satisfiability Modulo Theory (SMT) logical engine. Thus, the reasoner computes a formula that is the equivalent of the observation in logical terms as follows:

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

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 $\rho \coloneqq \mathbf{true}$. The logical expression $\kappa \coloneqq \sigma \wedge \rho$ represents all the information that the reasoner knows about the current CPS state s, i.e. that $s \in \mathcal{S}(\kappa)$.

4.3.1. Criticality Detection

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 $\mathcal{S}(\kappa) \cap \mathcal{S}(\phi)^C = \emptyset$. This is equivalent to checking whether the formula

$$\kappa \wedge \neg\phi \text{ is unsatisfiable.} \tag{6}$$

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

$$\kappa \wedge \phi \text{ is unsatisfiable.} \tag{7}$$

3. If both formulae in (6) and (7) 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 and proximity of the current state respectively, as explained in Section 4.3.2.

In the third case κ does not contain enough information to discriminate whether the CPS is in critical 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 refinement ρ.

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, 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 provides 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 sub-formulae. 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, thus it is crucial that the size of γ is as small as possible.

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[1] of a possibly simpler expression equivalent to α. Since formulae $\kappa \wedge \neg\phi$ and $\kappa \wedge \phi$ are mutually unsatisfiable, the assisted check can be defined as

$$\gamma \coloneqq interpolant(simplify(\kappa \wedge \neg\phi), simplify(\kappa \wedge \phi))$$

4.3.2. Predictiveness: State Criticality and Proximity from Conditions

In this section we define a notion of distance from a critical condition ϕ. Given a set X, a function $d: X \times X \to \mathbb{R}$ 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 $d: X \times X \to \mathbb{R}$ is called a *metric* if for all $x, y, z \in X$: (i) $d(x,y) = 0$ 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 $v \in \mathcal{V}$ and their observation times $v.t$, the set of states $\mathcal{S}$ can also be seen as a vector of $\mathbb{R}^{2n}$, where n is the number of variables. Thus, any metric d on $\mathbb{R}^{2n}$ 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.

Table 3.Example of metrics.

	$m_V(s,t)$	$= \sum_{v \in V} \lvert s(v) - t(v) \rvert$	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 $v_{\min}, v_{\min} \in \mathbb{R}$)
on	$h_V(s,t)$	$= \#\{v \in V \mid s(v) \neq t(v)\}$	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

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

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

$$D(\mathcal{S}(\kappa), \mathcal{S}(\phi)) = \inf_{\substack{s \vDash \kappa \\ t \vDash \phi}} d(s,t)$$

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

[1]As a reference, our prototype uses Z3 [33] with the tactic `(then simplify ctx-simplify ctx-solver-simplify)`.

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 3
shows 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.

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 [33] and by OptiMathSat [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

$$\begin{aligned} C_{\min}(\kappa,\phi) &:= \begin{cases} -D_{\max}(\kappa \wedge \neg\phi, \phi) = -\sup\limits_{s \vDash \kappa \wedge \neg\phi} \inf\limits_{t \vDash \phi} d(s,t) & \text{if } \kappa \wedge \neg\phi \text{ is satisfiable} \\ D_{\min}(\kappa \wedge \phi, \neg\phi) = \inf\limits_{s \vDash \kappa \wedge \phi} \inf\limits_{t \vDash \neg\phi} d(s,t) & \text{otherwise} \end{cases} \\ C_{\max}(\kappa,\phi) &:= \begin{cases} D_{\max}(\kappa \wedge \phi, \neg\phi) = \sup\limits_{s \vDash \kappa \wedge \phi} \inf\limits_{t \vDash \neg\phi} d(s,t) & \text{if } \kappa \wedge \phi \text{ is satisfiable} \\ -D_{\min}(\kappa \wedge \neg\phi, \phi) = -\inf\limits_{s \vDash \kappa \wedge \neg\phi} \inf\limits_{t \vDash \phi} d(s,t) & \text{otherwise} \end{cases} \end{aligned} \tag{8}$$

The meaning of previous definition is explained in, Table 4, which summarises the possible combinations of values of $C(\kappa, \phi)$, as a result of the logic in Figure 4 and definitions (8).

Table 4. Meaning of the results of the Reasoner.

$C_{\min}$	$C_{\max}$	$\kappa \wedge \neg\phi$	$\kappa \wedge \phi$	**Meaning**
negative	negative	sat	unsat	State is non critical regardless unobservables. $-C_{\min}$ and $-C_{\max}$ are the best and worst proximity values to ϕ
negative	positive	sat	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	unsat	sat	State is critical regardless unobservables, $C_{\min}$ and $-C_{\max}$ are the worst and best criticality values (i.e. proximity to $\neg\phi$)

$C_{\max}$ and $C_{\min}$ can be positive, negative, or zero. 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. Zero means the state is on the border of the critical states set.

Figure 5 shows the pseudo-algorithm 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 $\mathbb{R}^{2n}$ of choice. This enables to handle expressions $\kappa \wedge \phi$ and $\kappa \wedge \neg\phi$ easily without variable clashes and to minimise the distance at the same time.

```
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 PROXIMITYRANGE($\kappa$, $\phi$)

 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
    (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
      Cmin ← -DMAX(solver)
      solver.remove($\kappa_t$).assert($\kappa_s$)
      Cmax ← DMAX(solver)
  return (Cmin, Cmax)
function DMAX(solver)
  model ← solver.get-model()
  repeat
    dmax ← model.getvalue($\delta$)
    solver.assert($\delta >$ dmax $+ \varepsilon$)
    model ← solver.check-sat()
  until model is found
  return dmax
```

Figure 1. Proximity range pseudo-algorithm.

5. EXPERIMENTAL RESULTS

This section describes our prototype of the framework and the chemical process simulation testbed to prove the feasibility of the approach and the first performance results.

The prototype is based on Docker [35] 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 [36] docker container used to simulate[2]:
    a. The physical simulation of pumps and liquid flows, developed in the Typescript language.

[2]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 [37] library to send real Modbus messages on the network. The Observer used TShark/Wireshark [32] for traffic capturing and the same timeseries databases to store parsed messages. The real deep packet inspection for CPS, already established in literature [28], [30], 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.

b. The HMI implementing the manual control of the process, developed using Typescript functions and the Node-RED visual language Figure 6 shows a screenshot.

c. The automatic control, emulating the SCADA server, developed in Typescript and Node-RED.

d. The attacker's read and write commands to PLCs to emulate the scenarios in Section 3.

- **The Observer**: Modbus-like network messages from the chemical process simulations are stored in timeseries database with 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 [38] with its native query language and DataFrame files queries using the Pandas library [39], [40].

- **The Reasoner**: the prototype of the core of the proposed framework, developed in Python using Microsoft Z3, an open source SMT prover [33]. It implements the concepts described in Section 4 and provides the first performance and feasibility measurements. Results are store in timeseries databases for easy access. The reasoner is also instrumented with performance measurements using Prometheus [41].

- **Monitoring Interface**: Grafana [26] and Chronograf (part of the InfluxData suite [38]) containers that provide mature data visualisation and query interface to time series databases.

The whole prototype works on an Intel Core i7 laptop with 8 GB or RAM. Figure 7 shows the performance results of our benchmark. Each test generates random critical conditions based on a different number of variables up to 200. Then it generates a random CPS state. We performed 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.

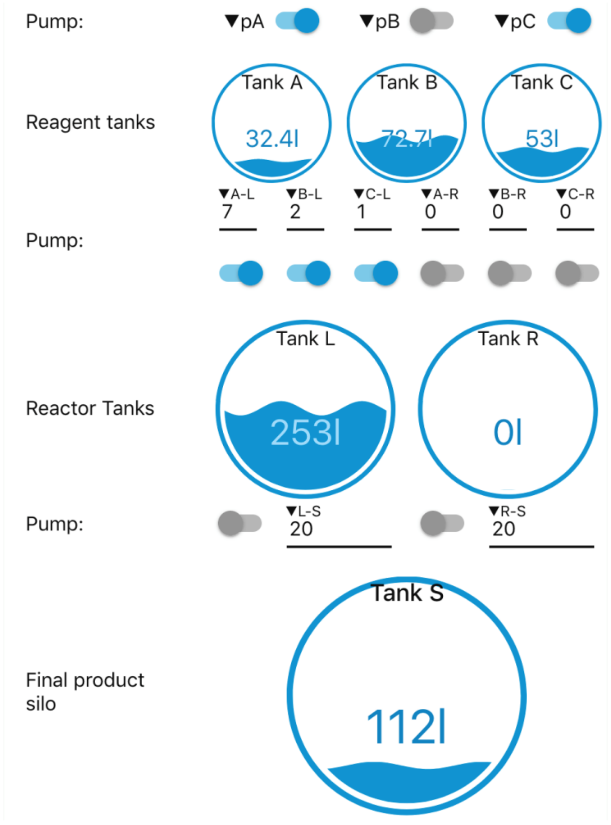

Figure 6: Screenshot of the emulated HMI.

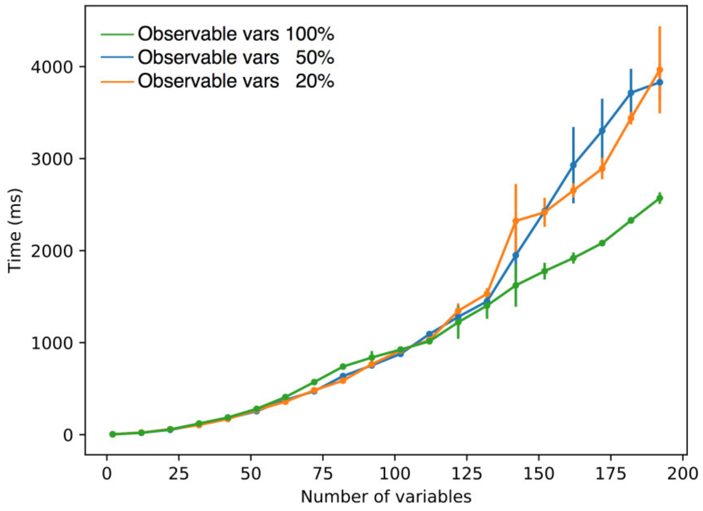

Figure 7: Computation time of $C(\kappa, \phi)$ from random benchmark.

6. Final Remarks

This work presents a specification-based predictive cyber security monitoring framework for cyber physical systems and improves previous works [5] and [6]. It enables specifying known critical conditions, through an easy but expressive formal language, that can be detected at runtime. It defines a quantitative notion of criticality of the CPS current state from the specified critical states: checking how the criticality changes in time enables security operators to predict whether the system is evolving towards critical states and how close it is from them, or similarly if it is returning to a licit state.

The novelty of the approach 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 to build a representation of this knowledge that correctly approximates the actual state of the system. Present work provides a way to specify critical conditions also in terms of constraints on the observation times, not only on their value. This provides a way to specify simple but effective temporal properties of the observed CPS behaviour.

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 criticality of the CPS. Our working prototype plots how the range changes in time, providing an overview of the evolution of the system w.r.t. the specified critical conditions which can be used as a criticality dashboard of support to Security Operaton Centers (SOC) and cyber incident response teams.

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 prove an expressive specification language and an efficient reasoning engine. While first results seem feasible and promising, further experiments can be performed to characterise critical conditions and will be the subject of further investigation to assess the limits of our approach.

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] A. Coletta, "Predictive Detection of Known Security Criticalities in Cyber Physical Systems with Unobservable Variables," in 11th international conference on security and its applications (cnsa), 2018, pp. 61–77.

[7] 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.

[8] 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.

[9] 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.

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

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

[12] 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.

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

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] D. Carasso, Exploring Splunk. CITO Research, 2012.

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

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

[23] J. Turnbull, The Logstash Book. James Turnbull, 2013.

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

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

[26] Grafana Labs, “Grafana.” 2017.

[27] LogRhythm Inc, “LogRhythm security intelligence and analytics platform.” 2017.

[28] I. Nai Fovino, 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.

[29] A. Carcano, A. Coletta, M. Guglielmi, M. Masera, I. Nai Fovino, and A. Trombetta, “A Multidimensional Critical State Analysis for Detecting Intrusions in SCADA Systems,” IEEE Transactions on Industrial Informatics, 2011.

[30] I. Nai Fovino, 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, Springer Berlin Heidelberg, 2011, pp. 116–127.

[31] “MODBUS Application Protocol Specification V1.1b3,” 2012.

[32] Wireshark Foundation, “Wireshark.” 2017.

[33] 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.

[34] R. Sebastiani and P. Trentin, “OptiMathSAT: A Tool for Optimization Modulo Theories,” in International conference on computer aided verification, 2015, pp. 447——454.

[35] Docker Inc, “Docker.” 2017.

[36] JS Foundation, “Node-RED.” 2017.

[37] G. Collins, “Pymodbus 1.2.0.” 2017.

[38] InfluxData Inc, “InfluxDB.” 2017.

[39] W. McKinney, “Data structures for statistical computing in python,” in Proceedings of the 9th python in science conference, 2010, pp. 51–56.

[40] W. McKinney, Python for data analysis: Data wrangling with pandas, numpy, and ipython. " O’Reilly Media, Inc.", 2012.

[41] P. Authors, “Prometheus-monitoring system & time series database.” prometheus. io, 2017

Authors

Alessio Coletta has 10+ years R&D experience in cyber security of Industrial Control Systems (ICS), working at the Joint Research Centre of the European Commission in the Security of Networked Critical Infrastructures unit, at the Global Cyber Security Center in Rome, in the Incident Prevention and Management unit of Poste Italiane, and currently at Magneti Marelli. He is a PhD candidate at the University of Trento (Italy) and Foundation Bruno Kessler (FBK, Trento). He holds a Master degree in Information Security at the Royal Holloway University of London and a Master degree in Computer Science at the Scuola Normale Superiore of Pisa.