IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 28 September 2022, accepted 4 October 2022, date of publication 10 October 2022, date of current version 26 October 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3213080

RESEARCH ARTICLE

Exploiting the Temporal Behavior of State Transitions for Intrusion Detection in ICS/SCADA

GORBY KABASELE NDONDA AND RAMIN SADRE, (Member, IEEE)
Institute for Information and Communication Technologies, Electronics and Applied Mathematics (ICTEAM), Université Catholique de Louvain (UCLouvain), 1348 Ottignies-Louvain-la-Neuve, Belgium

Corresponding author: Gorby Kabasele Ndonda (gorby.kabasele@uclouvain.be)

This work was supported in part by the SPW Recherche Program of Wallonia, Belgium, through Win2Wal Project PrEDICT under Grant 211012.

ABSTRACT Industrial Control Systems (ICS) monitor and control physical processes. The security of ICS has drawn the attention of many researchers since successful cyber-attacks against ICS can cause extensive damage in the physical world. Most of the existing literature describes solutions to protect an ICS against attacks directly targeting its underlying IT infrastructure. However, there are comparatively less works that focus on detecting cyber attacks against the physical process itself. Detection mechanisms that do so are said to be process aware. In this paper, we propose a time-based process aware intrusion detection system (IDS) that detects attacks against a physical process by leveraging its regular nature and temporal properties. The IDS learns the temporal behavior of the process variables and uses it to detect attacks. We evaluate the performance of our IDS on a public SCADA dataset and on a simulated SCADA system developed as part of this study, and we compare it with two other process-aware IDS proposed in the literature. The results show that our solution is able to detect attacks that are not detected by IDS that ignore temporal properties.

INDEX TERMS Anomaly detection, ICS, IDS, intrusion detection, process awareness, SCADA, temporal modeling.

I. INTRODUCTION

Industrial Control Systems (ICS), such as Supervisory Control and Data Acquisition (SCADA) systems, were originally completely isolated and often proprietary environments with no connections to the outside world. This has changed and we can nowadays find ICS built from commodity components connected to external networks or even the Internet. As a result, ICS are now much more exposed to cyber-attacks, giving attackers the possibility to not only steal data or harm services but also to damage physical processes [1].

One of the challenges when designing security solutions for existing ICS is that ICS are often meant to run for years without longer interruptions and that their resource-constrained components cannot be easily updated or reconfigured. This significantly hinders the deployment of host-based security solutions found in other environments. For that reason, many intrusion detection systems (IDS) proposed in the literature take the approach of focusing on monitoring the communication between the various ICS components without requiring changes on the latter. Such IDS can operate on three different levels: (i) They can monitor general characteristics of the ICS network and its hosts (e.g., the number of exchanged messages), for example in order to detect flooding attacks [2]; (ii) they can monitor details of the messages exchanged between the control servers and field devices in order to detect abnormal or malformed messages or message sequences [3]; finally, (iii) they can be *process aware*, i.e., they look for any behavior violating the semantics of the physical process controlled by the ICS [4], [5], [6]. IDS operating on level (i) and (ii) have the advantage that they are relatively general and do not require knowledge of the physical process. However, they cannot detect attacks specifically designed to damage the physical process, for example attacks that issue well-formed but undesired command sequences to actuators. In contrast, process aware IDS monitor *process variables*, i.e., the sensor values and the actuator states in the physical process.

Among the popular approaches for process aware IDS, two broad classes can be distinguished. Specification-based

The associate editor coordinating the review of this manuscript and approving it for publication was Leandros Maglaras.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

IDS compare the behavior or state of the monitored physical process to a specification provided by the process designer. Such IDS work especially well for processes where a formal specification is available or can be easily obtained. A typical example are power grids, for which the physical laws governing the power flows are well understood [7], [8], [9]. However, in this paper, we are interested in ICS where a formal specification or digital twin is not available for various reasons, for example because the physical process grew ''organically'' over its life time without its documentation being updated. For such processes, IDS designs have been proposed where the specification is *learned* from historical data. Again, two major categories can be identified among those IDS found in the literature: The first one leverages learned predictive models to predict the values of the process variables and the IDS raises an alert if the difference between the predicted value and the measured value is greater than a threshold [5], [10]. The drawback of this approach is that it does not fit well to the noisy nature of real sensors which makes comparing values challenging. Additionally, there is no formal way or algorithm to correctly select threshold. The second category uses invariant rules that must be always satisfied [4], [11], [12]. The drawback of invariant-based IDS is that they treat the states of the physical process without a notion of evolution in time. An attacker can evade detection by exploiting this limitation, for example by slowing down or permanently locking the physical process in a valid state, as demonstrated in this paper.

To overcome the above limitations of existing solutions, we present in this paper a process aware IDS that leverages the temporal characteristics of the system states. The IDS compares the current behavior of the ICS with a learned model that describes allowed transitions between critical values of the process variables and their time properties. Our main contributions are:

- We propose a time-based IDS which allows to detect process oriented attacks. The detection algorithm learns its model by observing the process variables of the ICS during a training phase. With this approach, we target processes for which a formal specification of their behavior is not available or difficult to obtain.
- The detection algorithm depends only on a few configuration parameters provided by the operator.
- We demonstrate its performance on two case studies, one from a real-world ICS testbed and one from a simulated environment, and compare it to two learning-based invariant-based and prediction-based IDS proposed in the literature.

The structure of the paper is as follows. Section II presents important concepts used in the remainder of the paper. Section III discusses related work. Section IV describes our detection approach. Its evaluation is presented in Section V. Limitations are discussed in Section VI. The paper is concluded in Section VII.

II. IMPORTANT CONCEPTS

A. INDUSTRIAL CONTROL SYSTEMS

ICS are systems to control and manage industrial processes occurring in industrial facilities like factories, plants, or water treatment facilities [1]. An ICS typically consists of one ore more Supervisory Control and Data Acquisition systems (SCADA) or Distributed Control Systems (DCS) which contain at least two types of components: a Master Terminal Unit (MTU) and distributed controller devices. An MTU is a server in charge of monitoring and controlling a physical process in a plant. It gathers sensor and status data from the field devices, takes decisions when certain events happen, and sends commands to them. The field devices are Programmable Logic Controllers (PLC) or Remote Terminal Units (RTU) that are connected to the actual sensors (temperature sensors, pressure sensors, etc.) and actuators (engines, valves, etc.). In modern ICS, the MTU and the field devices communicate over an IP-based network, using protocols such as Modbus [13]. ICS are designed to run autonomously, but the MTU is usually connected to a Human Machine Interface (HMI) so that operators can set objectives or override automatic decisions.

B. PROCESS VARIABLES

A process variable stores a certain value of a physical process, e.g., the level of water in a tank or the flow rate of a pipe. The state of a process at a given time is determined by the values of its process variables. The latter can be divided into two categories. *Discrete variables* have a small set of possible values. An example is a variable representing the state of a valve: ''Open'' or ''Closed''. *Continuous variables* have a more or less continuous value range as the range of values depend on the accuracy of the sensors. They usually represent physical properties measured by sensors, like the water temperature.

The ICS controls the process by manipulating actuators depending on sensor values. For example, once the water level in a tank has reached a certain threshold, the ICS may decide to open a valve. We refer to these values as *critical values*. Consider the physical process in Figure 1. The goal of this process is for *Tank 3* (T3) to release 60 units of a product C. T3 has a maximum capacity of 80 units. The tanks *Tank 1* (T1) and *Tank 2* (T2) contain respectively product A and B. First, T1 releases 20 units of product A, and then T2 release 40 units of product B, so they can be mixed in T3 and create product C. There are six process variables in this example: one discrete variable for each valve and one continuous variable for each tank's level. The ICS has to open the different valves according to the level in T3, therefore there are three critical values for T3's level that trigger a change in the valve states: 0, 20, 60. It is important to repeat here that actuators also have critical values. For example for a valve, there are two critical values, open and closed.

In a process-oriented attack, the attacker tries to disturb the physical process. For example, an attacker could send a command over the network to open the valve of T3 while

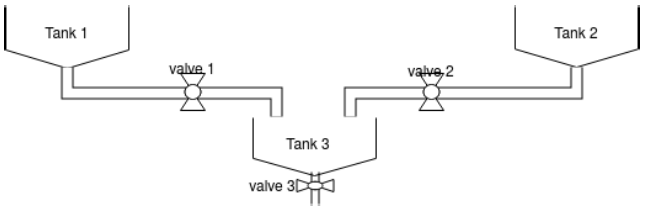


FIGURE 1. Simplified three tank system.

the tank is being filled [14]. An invariant-based IDS (see Section III) can detect this attack because it violates the invariant of the process stating that the valve of T3 is always closed when the tank is being filled. However, the attacker could also wait until the tank is empty and then prevent the valve from closing, for example by overwriting the `Close` command sent by the ICS. An empty tank with an open valve is a legitimate state of the system; it happens for a short time at the end of emptying the tank. The key here is the time aspect. Our IDS is based on the observation that critical values (and the transitions between them) are associated with a certain duration.

III. RELATED WORK

Attacks against ICS can be classified by the intent and target of an attacker. If the cyber-part of the ICS is the target of an attack, like for DDoS attacks, an IDS that uses network information can be appropriate. Such an IDS detects attacks by monitoring network traffic and inspecting packet headers/payload of ICS protocols such as Modbus/TCP [13]. There exist several works in the literature to detect such attacks [3], [15], [16].

Process-oriented attacks are different as they target the physical process controlled by the ICS. This requires a certain understanding of the process by the attacker in contrast to network attacks. IDS that are aware of the physical process and use that knowledge to detect attacks are called *process aware*. They usually belong to the class of anomaly-based IDS, i.e., they monitor a system and compare its activities with a model describing its expected behavior. Process-aware IDS are the focus of our work, so we limit the works presented in this section to those IDS.

In [17], Markov chains are used to detect malicious sequences of commands sent to actuators and sensors, denoted as *sequence attacks*. States in the chain represent events, for example the response of a field device to a request from the server and the transitions are actions (e.g., Modbus function calls) leading from one state to another with a certain probability. The Markov chain is learned during a training phase from network communications between the control server and the field devices. At runtime, the IDS compares the current system behavior with the stochastic behavior described by the Markov chain. This approach is sensitive to the training data. If the training set is too small, it can result in over-fitted models that do not tolerate small deviations that can be often found in real systems (the problem of naturally occurring deviations has been addressed in the design of an IDS described in [18]). If the training set is too large, a very general model is obtained where most transitions have a non-zero probability.

Koucham et al. [11] also propose an IDS to detect sequence attacks. The IDS takes specifications given by *Linear Temporal Logic* formulas that state the natural order of events in the physical process. The formulas are generated from network traces and then filtered to reduce their number. However, as the authors acknowledge, with the number of specifications they have, when an attack occurs, there are so many specifications that are violated that it is challenging to do a precise analysis. Also, the IDS has to keep a subset of the current trace to see if a specification is violated or not. A large subset has more chance to detect attacks but requires more resources, while a smaller set may miss attacks.

Adepu et al. [19] propose an invariant-based IDS. The invariants are rules that must be satisfied at all times. The drawback of this approach is that it requires an accurate knowledge of the physical process to be effective. Feng et al. [4] mitigate this drawback by presenting a framework based on data mining to generate the invariants of a physical process from datasets and log files collected at the ICS. It generates few false positive but it can detect an attack only if the latter impacts the system in such a way that it violates an invariant. In Section V, we show the limitations of the approach in certain situations.

Menzel et al. [20] takes a similar approach to secure SCADA systems for distribution grids. The author presents a formal model representing an electrical grid (generators, producers, power lines,...) and they define both physical and safety requirements that are checked on the model. A violation of those requirement indicates an attack.

Monzer et al. [21] propose a model-based IDS which encompasses both the architecture of the ICS and the physical process. They model the PLCs behavior via an Hybrid Automata model and present a model of the physical process based on its inputs (e.g commands), its outputs (e.g measurements) and the physical law of the process. From this model they generate rule for an open source IDS (e.g Zeek) that will feed the model from the network traffic and detect attacks.

Compared to the work presented here, the requirement are manually designed.

While [19] and [4] try to identify properties that always hold, Sicard et al. [22] look for system states that should never happen. They implement a filter placed between the PLC and the attached actuators and sensors. First, they identify the system states that are critical, i.e., the ones that can damage the system. Then, they create a process model using Deterministic Finite Automata, which describes all possible states and actions of the system. In addition, they create a control model using Petri Nets, representing the scheduling of the actions taken by the system to achieve its goal. To detect attacks, they use the two models to calculate the distance between the current system state and the critical states. If the distance to the closest critical state is below a defined threshold,

commands sent by the PLC to the actuators are immediately blocked. We argue that being able to trace the commands exchanged between a PLC and the actuator/sensors would require significant modifications in the field devices used in practice. Secondly, the blocking is completely hidden to the control server, meaning that it cannot differentiate between blocked commands and commands not correctly executed due to a malfunction. Similarly, Carcano et al. [23] describe a formal language to declare critical system states and define a distance function to detect whether the current state of the system will reach a critical state in the near future. Unfortunately, they do not provide an experimental evaluation, but like other invariant-based approaches, their approach suffers from the fact that attacks causing damage to the system without leading to a critical state cannot be detected.

The above approaches do not consider time, in the sense that they do not reason on how long a system should stay in a certain state or how long a transition between two states should take. There exists work that considers time in their detection process [24], [25], [26], [27]. For example, Lin et al. [24] and [25] leverage temporal properties of SCADA systems to detect attacks. However, their approach focuses on the temporal properties of events at the network traffic level, i.e, when network messages are sent or received. Those work are effective in detecting network attacks such as drop, replay or delay attacks. However, they are not looking at the behavior of the system at process level, i.e., they are not process aware and cannot detect process-related attacks. Yang et al. [27] present an IDS exploiting the correlation between the sensor readings and the commands sent by the system to identify set-points. The correlation is learned from logs and represented as a control graph where the nodes can be set-points, sensor readings, or commands, and the edges are weights describing the correlation between two nodes. During detection time, the IDS uses a sliding window to monitor the evolution of the correlation between the nodes, specifically between the sensor readings and the commands, to detect anomaly. When the computed correlation deviates too much from the expected one repeatedly according to a predefined threshold, an alert is raised. In that approach, time is implicitly considered because of the sliding window. This IDS is able to detect attacks rewriting the sensor reading in an obvious way, i.e., when the reading does not make sense in the state in which the process is, or in a stealthy way where the readings sent by the attacker are just the repetition of what the attacker previously observed. The performance of the IDS highly depends on the size of the sliding window and the value of the threshold. They impact the number of attacks detected and the number of false alerts. However, in order to choose those values, a priori knowledge of the process is required. No algorithm to select those values is provided.

The works in [10] and [28] also use prediction-based detection with a threshold controlling the level of alerts raised. In both those work, a Cumulative Sum algorithm is used to limit the number of alerts raised by waiting for consecutive deviations (residuals) of the system from the expected behavior. The authors leverage the Linear Dynamical State-Space model (LDS). LDS models a subset of the state space models. For the inputs (control commands u_k) the outputs (sensor measurements y_k) and the state x_k of a physical process, the dynamic modeling of process is

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k + \epsilon_k \\ y_k &= Cx_k + Du_l + e_k \end{aligned}$$

where A, B, C and D are modeling metrics of the dynamics of the physical process, and e_k and ϵ_k are sensor and perturbation noise. In order to perform predictions, a deep understanding of the dynamics of the physical process is required to correctly use the LDS model. The approach followed in our paper requires only little information about the physical process and only needs the measurements and their origin, i.e., from which actuator or sensors they come.

Table 1 gives a summary of the related work. Note that it only includes the works related to the detection of process-oriented attacks and is not exhaustive.

In the next sections, we describe our time-based intrusion detection system and demonstrate that it is able to detect attacks that would not be noticed by approaches that only look at the system state.

IV. TIME-BASED INTRUSION DETECTION SYSTEM

We present our time-based IDS that exploits temporal properties of process variables. The IDS learns these properties for a physical process from trace files that were recorded, for example, at the control server of the ICS (or in the network if the messages exchanged between the ICS components are not encrypted [13]). Once the temporal properties have been learned, the IDS can move to the detection phase where it operates on live data obtained from the same source. Our solution has been designed for ICS/SCADA systems where a centralized server monitors and controls the physical process.

A. SCOPE AND ASSUMPTIONS

As already explained in Section I, we assume that a formal specification of the physical process is not available and therefore has to be learned.

We assume that process variables move between critical values. We call the event of a variable moving from one critical value to another a *transition*. For a discrete variable, like the state of a switch, a transition is fast, but for a continuous variable it can take several minutes or hours depending on the underlying physical process. We assume that process variables behave consistently in their transitions throughout the lifetime of the physical process. In other words, we assume that critical values will be recurrently visited by the corresponding variable. We refer to those variables as *recurrent variables*. Such variables hold specific values when there is a change in the process state handled by the system. A recurrent variable does not necessarily need to follow a fixed order when passing through a series of critical values — only the fact that one or more critical values are

TABLE 1. Summary of related works on the detection of process-oriented attacks. Data-driven: "yes" means that the model and/or its parameters are learned from data traces.

Related work	Data-driven	Data source	Technique
Caseli et al.[17]	Yes	Network	Markov Chain
Koucham et al.[11]	Yes	Network	Linear Temporal Logic
Menzel et al.[20]	No	Logs	Formal Model
Monzer et al.[21]	No	Network	Automata& Formal Model
Adepu et al.[19]	No	Logs	Invariants
Feng et al.[4]	Yes	Logs	Invariants-Mining
Sicard et al.[22]	Yes	Logs	DFA and Petri Nets
Carcano et al. [23]	No	Unknown	Invariants
Yang et al. [27]	Yes	Logs	Correlation Control Graph

recurrently visited matters. It should be noted that not all process variables behave like this. Figure 2 shows examples of the two types of variables. The recurrent variable in Figure 2a moves between specific values, although not necessarily with constant time between the changes. The decreasing and the increasing phases occur because the variable has reached a certain value causing a change of state. The variable shown in Figure 2b does not depict such a behavior. Our IDS can only monitor those variables in an ICS that behave like the one in Figure 2a.

Furthermore, we assume that actuators are represented by discrete variables that hold their state as set by the ICS. That does not necessarily mean that an actuator cannot have a continuous physical state. Obviously, for example, the real speed of an engine is not a discrete quantity. Rather, it means that the ICS sets the target speed of the engine in form of discrete values.

We also assume that if an attacker is tampering with a process variable, it will have an impact on how fast it moves between its critical values or on other process variables. It may take more or less time than expected or a transition happens that was not observed during the learning phase.

Finally, we assume that the learning dataset is of sufficient length that means all recurrent variables have visited all their critical values several times during the measurement period. Because we follow a data-driven approach, a representative trace is required, which is an obvious limitation inherent to all data-driven approaches, however getting such a trace is feasible in ICS environments as traces are consistently stored for different reasons, e.g., diagnostics or quality assurance.

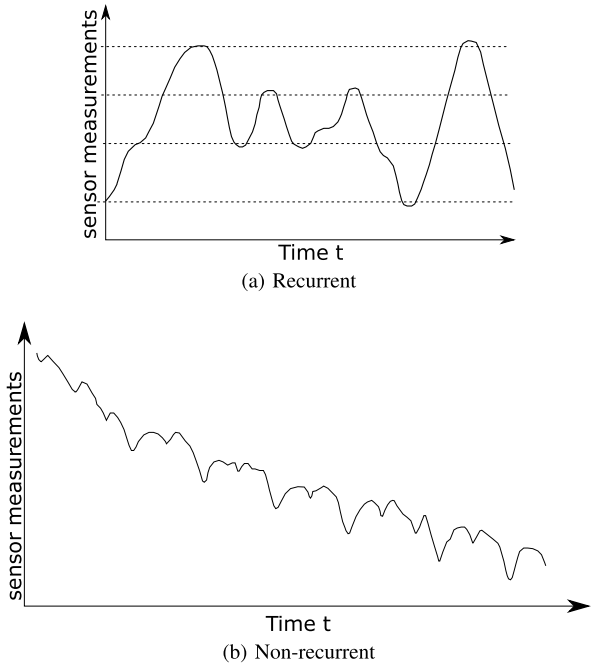


FIGURE 2. Types of process variables.

B. ATTACK MODEL

In the following, we describe the attack model considered in this paper. An attack model defines which part of the system is targeted by an attacker, what are their goals, and what are their capabilities. We will only consider attacks targeting the physical process. The goal of the attacker is to cause damage by manipulating the process variables such that the normal operation of the process is disturbed. For example, the attacker could prevent a tank from emptying and thus cause an overflow. We assume a scenario where an attacker is inside the ICS network:

- The attacker can connect to the PLCs in the network.
- The attacker can read and write the registers of a PLC by sending requests to it.
- The attacker knows the physical process being controlled by the ICS/SCADA, i.e., the attacker knows the explicit relations and dependencies existing between the process variables and exploits that knowledge. By explicit dependencies, we mean the dependencies explicitly defined in the design of the process.
- The attacker knows which process variable is handled by which PLC.
- The attacker cannot forge the response of the PLCs. That means that it can not control the response sent by a PLC upon receiving a request from the control server.
- The attacker does not communicate with the control server.

Given that model, attackers can damage a physical process efficiently. They have enough knowledge to identify the process variables that have to be to manipulated in order to achieve their goal. Since they know the mapping between those variables and the PLCs, they can send commands to them to control the process according to their goal. In this model, the control server and the system logs remain a

TABLE 2. Summary of assumptions and attack model.

Aspect	Assumption
Process variable type	continuous or discrete
Process variable behavior	recurrent moving between two or more critical values time to complete transition is consistent
Actuators	represented by discrete variables
Learning dataset	sufficient length (all critical values visited several times)
Attack target	physical process
Attacker knowledge	knows the physical process knows which process variable is handled by which PLC
Attacker capabilities	can connect to PLCs can send read/write requests to PLCs cannot forge PLC responses does not communicate with the control server the control server and system logs remain reliable

reliable source of information and they will be used by our time-based IDS to detect process oriented attacks.

Table 2 summarizes our assumptions on the system behavior and the attack model.

C. LEARNING PHASE

The learning phase of the IDS expects an attack free trace of the system. The trace is a series of snapshots of the process variables taken every second (this is a typical update frequency in ICS [4]). In addition, we tell the IDS which variables are discrete and which are continuous, and which discrete variables are representing actuators.

Analyzing the relationships between potentially hundreds of process variables is a complex task. We follow an efficient approach with a minimum number of configuration parameters in the following. The IDS performs two major steps during the learning phase (Figure 3):

1) STEP 1: IDENTIFYING CRITICAL VALUES

For a discrete variable, the critical values are the values that it can take. In particular, an actuator corresponds to a discrete variable a with critical values $C_a = \{c_1^a, \ldots, c_n^a\}$. For a continuous variable s holding a sensor value, its critical values are those values that cause the ICS to change the state of one or several actuators.

Let a be an actuator variable and s be a continuous sensor variable. For a transition $c \to d$ from critical value c to d of a happening at times $t^1, \ldots, t^m$ in the learning dataset, we look at the values $v_1, \ldots, v_m$ that s takes at those times. If a certain sensor value v is linked to the transition $c \to d$, we should see $v = v_1 = \ldots = v_m$ and we conclude that v is a critical value of s (in theory, an actuator transition could be linked to two or more critical values of the same sensor s, but we have not seen that happen in practice). According to [4], sensor values can be perturbed by noise following a doubly truncated normal distribution with mean 0, therefore if μ is the mean of $v_1, \ldots, v_m$, we should observe that all v_i fall in a bin $[\mu - \frac{\Delta}{2}, \mu + \frac{\Delta}{2}]$ for some bin width Δ.

We repeat this for all transitions of all actuators and obtain for s a set of bins. We expect that the widths of some bins are relatively small for those actuator transitions where s has a critical value and relatively large when the value of s is not related at all to the transition. To identify to which of those two groups each bin belongs, we normalize the bin widths by the respective bin means and use *Mean Shift* [29], a non parametric clustering algorithm for mode seeking. We then only keep the bins assigned to the cluster with the smallest mean bin width. Those selected bins $C_s = \{[l_1, r_2], \ldots, [l_{c_s}, r_{c_s}]\}$ represent the c_s critical values of s. We say that s has reached its ith critical value if its current value v falls in the range $[l_i, r_i]$. If multiple bins contain v, the closest is selected. Note that we can see discrete variables as variables where all the bins only contain single values, i.e., $l_i = r_i$. For continuous variables, we never let the bin width go below six standard deviations of the values observed in the learning data.

For efficiency reasons, we then discretize all bin boundaries to multiples of the smallest bin size in C_s. This introduces a small error, but allows to implement the check whether a sensor has reached a critical value as a table lookup.

TABLE 3. Symbols used in the description of our IDS. See section IV for a detailed explanation.

Symbol	Description
C_a	Set of critical values of discrete variable a
C_s	Set of critical values of continuous variable s
$a \to b$	Transition from critical value a to b
$v_1, v_2, \ldots$	Series of values observed for a process variable
(t, u)	Observation of transition time t and update step u
$d(A, B)$	Euclidean distance between two points A and B
$d_k(A)$	Distance between point B and kth nearest neighbour
$rd(A, B)$	Reachable distance between two points A and B
$N_k(A)$	set of k nearest neighbours of point A
$lrd(A)$	Local reachability density of point A
$LOF(A)$	Local Outlier Factor of point A
$SDAM(X)$	Sum of squared Deviations for Array Mean of X
$O^p_{x\to y}$	Observations of $x \to y$ for variable p
$\theta^p_{x\to y}$	LOF threshold of variable p and $x \to y$
$O^p_{x,still}$	Still times of variable p at value x
$\theta^p_{x\to y}$	LOF thresholds of still times of p at x
$maxT_x$	Longest observed duration of transition from x
δ	Relaxation factor for test on $maxT$
min_p	Smallest observed value of variable p
max_p	Greatest observed value of variable p
ϵ	Tolerance for test on min_p and max_p

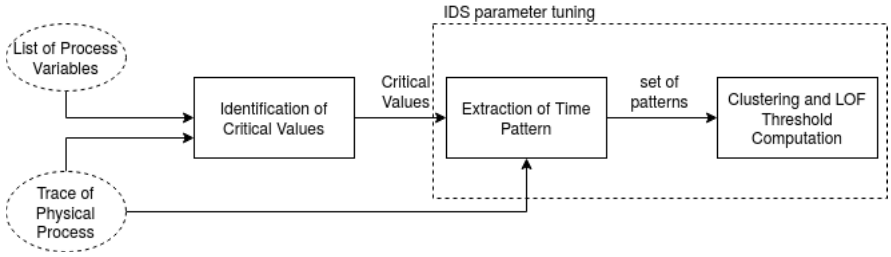


FIGURE 3. Learning phase steps.

2) STEP 2: TIME PATTERN EXTRACTION

Once the critical values of all process variables have been identified, we extract for each continuous variable p the temporal properties of the transitions occurring in the physical process. The series of values $v_1, v_2, \ldots, v_{n-1}, v_n$ observed for a process variable p form a transition if v_1 reaches some critical value x and v_n reaches a different critical value $y \neq x$ and the intermediate values $v_2, \ldots, v_{n-1}$ do not reach any critical value. We call n the transition time, i.e., the number of 1-second steps the transition took, and $u = \frac{v_n - v_1}{n-1}$ the mean update step.

Note that this means that the IDS is *not* trying to find periodic patterns in the evolution of a sensor. The IDS looks for the time needed to complete a transition from one critical value to another. The absolute timing of when the said transition occurs is not explicitly taken into account by the IDS since transitions are treated independently and the order in which they occur is not an information used by the IDS. By doing so, the IDS is not limited to variables that exhibit periodic patterns and that must run the patterns in specific amounts of time. Imagine the example in Figure 4 showing the evolution of a sensor variable. The variable has four critical values $c0, c1, c2, c3$. There are three transitions in increasing order, $c0 \rightarrow c1, c1 \rightarrow c2, c2 \rightarrow c3$ and three in decreasing order for a total of six transitions. Looking at the three transitions from $c1$ to $c2$, we can observe that they have different durations. Similarly, the two occurrences of the transition $c0 \rightarrow c1$ indicated in the figure take different amounts of time. This is the information that the IDS monitors explicitly, and not the time when a transition has started. Obviously, the absolute starting time of a transition depends on the transitions occurring before it.

In general, we observe for a process variable multiple occurrences (t_i, u_i) of a transition from critical value x to critical value y. The transition times t_i of those occurrences are not necessarily constant, as visible in Figure 4, and might vary randomly due to noise or depending on internal or external factors. During the detection phase (see below), the IDS will see a transition $x \rightarrow y$ occur with time t' and update step u' and will have to decide whether (t', u') is sufficiently similar to the occurrences (t_i, u_i) observed during the learning phase.

Unfortunately, this task is complicated by the presence of outliers. The latter can appear during the detection phase

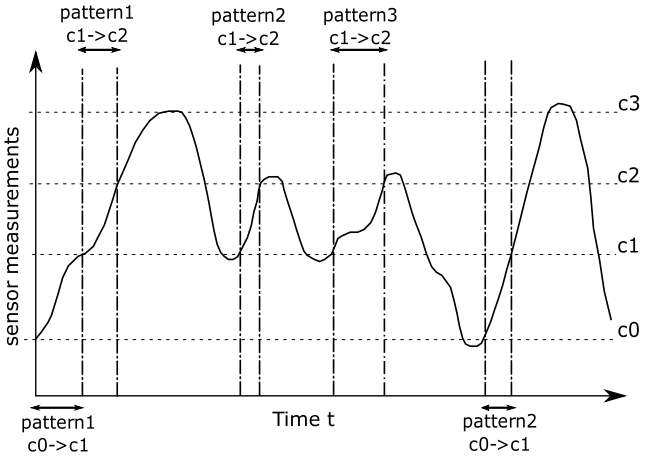


FIGURE 4. Time pattern example.

as well as during the learning phase. Most of those outliers are harmless in the sense that they are caused by noise and should therefore be accepted as legitimate. However, others are anomalous and should be ignored during the learning phase (e.g., because they are caused by unique incidents) and raise an alarm during the detection phase. The challenge is to find a distance function and threshold θ that allows us to tell apart anomalous outliers from legitimate data. To overcome this challenge, we use the unsupervised anomaly detection algorithm called Local Outlier Factor (LOF) [30]. It assigns to each data point a score representing the local density deviation with respect to its neighbors. Intuitvely, an outlier (or anomaly) is a data point with a lower density than its neighborhood. If clusters represent legitimately occurring values, outliers caused by noise are close to a cluster but are not part of it, while anomalous outliers are far from all clusters. By computing the density score, we can estimate a threshold for the score that determines whether a new data point is an outlier or not.

LOF computes for every point a ratio of its density compared to its k nearest neighbours. It uses a distance called the *reachability distance*. Let d be the Euclidean distance and d_k be a distance function such that for a point A, $d_k(A)$ gives the $d(A, C)$ where C is the kth nearest neighbours of A. The reachability distance rd between two points A and B is

$$rd(A, B) = max(d_k(B), d(A, B)) \quad (1)$$

The algorithm defines the local reachability density *lrd* as

$$lrd(A) = \left(\frac{\sum_{B \in N_k(A)} rd(A, B)}{|N_k(A)|} \right)^{-1} \quad (2)$$

where $N_k(A)$ is the set of the k nearest neighbours of A. The Local Outlier Factor of a point [30] is then defined as the average of the ratio of the local reachability density of its neighbours and its own:

$$LOF(A) = \frac{\sum_{B \in N_k(A)} lrd(B)/lrd(A)}{|N_k(A)|} \quad (3)$$

The higher the value, the more its density varies significantly from its k nearest neighbours. An outlier close to a cluster will have a lower LOF than an outlier far from every cluster. We compute the LOF of every data point and we split the resulting LOF values in two classes (outliers and non-outliers) using the Jenks Natural Break Classification Method [31]. This method is a one-dimensional classification method that aims at minimizing the intra-class variance and maximizing the inter-class variance. Given a set of data points $X = x_0, x_1, \ldots, x_n$ with mean $\mu(X)$ and such that $x_i \leq x_j$ for $i \neq j$, the method works as follows:

1) Compute the Sum of squared Deviations for Array Mean, $SDAM(X) = \sum_{i=0}^{n} x_i - \mu(X)$
2) Create all possible two-classes assignments $C = [(x_0), (x_1, \ldots, x_n)], \ldots, [(x_0, \ldots, x_{n-1}), x_n]$
3) For each assignment in C:
   - Compute the Sum of squared Deviation for Class Mean, $SDAMC = SDAM(C_{i0}) + SDAM(C_{i1})$ where C_{i0} is the set of points belonging to the first class in the i^{th} assignment. C_{i1} is the second class.
   - From all *SDAMC* computed, take the one with the minimum value, which indicates the class assignment with the lowest in-class variance.

Using this approach, we end up with each data point assigned to either the outlier class or the non-outlier class. By selecting the LOF score of the most deviating data point of the non-outlier data class, we can select a threshold θ such that any point with a LOF higher than θ is considered as an outlier. A datapoint x is considered as an outlier if $LOF(x) > \theta$.

To summarize, the time pattern extraction yields for each variable p and each of its transitions $x \rightarrow y$ with $x \neq y$ being critical values of p the list of observed occurrences $O^p_{x \rightarrow y} = (t_i, u_i)$, and the LOF threshold $\theta^p_{x \rightarrow y}$. If $O^p_{x \rightarrow y} = \varnothing$, it means that the transition does not occur.

We also define the *stillness time* for each critical value x of a discrete variable. It gives the time a variable stays in a critical value before moving to a non-critical or another critical value. Stillness times are particularly interesting for discrete variables representing actuator states, which often stay constant over extended periods, for example when the ICS waits for a continuous variable, like the tank level, to reach a certain threshold value. We also apply the clustering and LOF algorithm to them, however without the mean update step u component. The result is again a list of observed occurrences $O^p_{x,still}$ and a LOF threshold $\theta^p_{x,still}$.

D. DETECTION PHASE

During the detection phase, the IDS follows the values of all process variables. When it detects the occurrence of a transition $x \rightarrow y$ (as defined in the previous section) for a process variable p, it first checks whether $O^p_{x \rightarrow y}$ is empty. If it is empty, the transition is not allowed and an alert is raised. If it is not empty, it computes the transition time and mean update step (t', u') of the occurrence and $LOF(t', u')$ as if the new data point (t', u') was part of the data points observed during the learning phase. If the LOF is greater than the threshold $\theta^p_{x \rightarrow y}$, the occurrence is regarded as an anomalous outlier and an alert is raised. For discrete variables, the IDS also (a) checks whether the current value is a critical value, and (b) monitors the stillness times and compares their LOFs to the respective thresholds.

The IDS does not always have to wait until the end of a transition to perform a detection. For each process variable, the IDS continuously keeps track of the time since the begin of a transition from a critical value x and triggers the above detection process if it exceeds the maximum duration $maxT_x$ observed in the training phase for any transition from x. Similarly, the stillness times of actuator variables are continuously checked. How aggressively the observed duration should be checked against $maxT_x$ depends on the type of physical process to be monitored. In a sensitive physical process with strict time constraints, $maxT_x$ can be seen as a strict limit. However, in practice, many physical processes have relatively soft time constraints and being too strict would trigger a series of false alerts (one per time step) in the case of noise or incomplete training data. For such processes, our IDS allows to use a relaxed maximum duration $\delta \cdot maxT_x$ instead, with $\delta \geq 1$ being a configuration parameter set by the operator and with default value 1.

The above temporal analyses of transitions are the core of the detection process of our IDS. Nevertheless, it also performs other checks to detect other kinds of attacks. During the detection phase, the IDS checks for each variable p whether its current value v is within the range of values observed during the learning phase. The goal is to detect overflow/underflow attacks where values go beyond the process variable limits. Because of the noise, an exact comparison with the maximum learned value max_p (resp. minimum min_p) could lead to false alerts. We therefore check if $\frac{|max_p - v|}{max_p - min_p} \leq \epsilon$ or $\frac{|min_p - v|}{max_p - min_p} \leq \epsilon$ where ϵ is by default set to 0.01.

V. EVALUATION

A. METHODOLOGY AND DATASETS

We evaluate our IDS on two case studies: a process running in an ICS/SCADA testbed (SWaT process) and a process running in an ICS/SCADA simulation. Both come from the literature, were already used to evaluation Intrusion Detection

Systems in the past, and allow to test a wide range of attacks[1]. For each case study, there are two traces: an attack-free trace that we use for learning, and a trace containing normal operation and attacks that we use to test the performance of our IDS. A data point in a trace is a one-second snapshot of the state (i.e., the values of the process variables) of the physical process. For the attack trace, the job of the IDS is to classify each data point as malicious or legitimate. As usual, we denote correctly classified data points as true positives (TP) resp. true negatives (TN), and wrongly classified ones as false negatives (FN) resp. false positives (FP). We also define the detection rate $DR = \frac{TP}{TP+FN}$ and the false positive rate $FPR = \frac{FP}{FP+TN}$.

The attacks performed on the processes have different goals such as damaging a component of the process, or halting operations. However, we can identify three basic types of actions:

- Actuator modification: The attacker modifies the state of an actuator, for example opening a valve that was closed.
- Actuator blocking: The attacker prevents an actuator from changing state, for example leaving a valve closed when it is supposed to be opened.
- Sensor rewriting: The attacker overwrites sensor values.

The studied datasets contain 36 attacks (SWaT process), respectively 17 attacks (simulated process). Some of the attacks use more than one of the above three types of actions. The column ''Occurrence'' in Tables 8 and 15 shows how often a specific type of action occurred in the attack datasets of the two processes. The occurrence of an action covers the entirety of the period during which the modifications performed by that action are effective. Note that the length of that period will depend on the attack and can be longer than one second. An action counts as detected by an IDS if the latter raised at least one alarm during the activity period of the attack the action was part of. For a detailed description of the attacks on the SWaT process refer to [12].

The SWaT process is a fairly complex process such that we had to adapt how we counted false positives in our experiment. When an attacker launches an attack on a process variable, there might be a direct effect at the time of the attack, but depending on the attack and the implicit dependency relationships existing between process variables, other effects might begin to appear way after the activity of the attacker has ended. For example, one of the attacks in the trace still has an impact on the physical process two hours after the attack stopped. The complexity of the SWaT process makes it difficult to accurately pinpoint which after-effect is linked to which attack.

Therefore, we consider malicious states a bit differently than in [4], where the states of the physical process inside the time window of an attack, i.e., the time between the start and the end of the attacker's actions, are labelled as malicious. We also take this approach but we add the possibility that transitions outside of an attack window can be impacted as well. The reasoning is the following, the SWAT process operates like a pipeline where event occur in a specific order, for example when a tank is filled, a valve will be opened. So an attack on a process variable can impact the transition of an other one that depends on the targeted variable. As the impact can happen after the attack, we label the states containing such impacted transitions as malicious as there are a consequence of the attack. To identify these transitions, we use an heuristic which consist of looking of at all the transitions starting before an attack period and finishing at the end of the period. Among those transition, we only keep the unique ones for each recurrent process variable as the uniqueness indicates that these transitions are a result of the attacks.

We compare our IDS to a process-aware invariant-based IDS [4] and a prediction-based IDS using auto-regression [5]. We have chosen those two works for the comparison because:

1) They have been specifically designed to detect process aware attacks.
2) Similar to ours, those two works follow a data-driven approach where the detection model is automatically learned from the learning trace. Other approaches, such as [12], [19], and [23], create their models from the design specification of the physical process, which means that their performance is limited by how accurately the author of the specification has described the physical process.
3) They are representatives of two existing major approaches to detect process aware attacks, namely, invariant testing and prediction. We think that it is interesting to study the respective strengths and weaknesses of each approach.
4) Finally, we also selected those works because of their visibility in the scientific community.

The evaluation is performed on a Dell XPS 13 with 8 GB of RAM and a 2.70 GHZ CPU. We have implemented our timed-based IDS in Python. The invariant-based IDS has been implemented in two parts: one in Java for the mining of invariants and one in Python for the detection process. The prediction-based IDS has been implemented in Python.

In the following, we will abbreviate our time-based IDS as T-IDS, the invariant-based IDS as I-IDS, and the prediction-based IDS using autoregression as AR-IDS.

B. INVARIANT AND PREDICTION BASED IDS

Before presenting the results obtained for the two case studies in the next sections, we briefly describe the two reference IDS since a basic knowledge of how they work is important to understanding the results.

The IDS of Feng et al. [4] leverages the invariant properties of the process to detect attacks. The IDS creates invariant rules of the form $A \rightarrow B$, where A and B are predicates/conditions on the physical process. An invariant is

[1]Obviously, getting access to a live SCADA system is challenging as operators are not inclined to allow researchers to perform attacks on their systems.

violated when A is satisfied and B is not. There are two types of predicates: distributed-driven and event-driven. The first type considers changes in sensor readings to be determined by the current state of the physical process. The IDS assumes that each update of a sensor value follows one of K hidden Gaussian distributions. The distributions are found by fitting a Gaussian Mixture Model to the difference between each update of the values of a sensor. The predicate is satisfied for an observed sensor update if there is a high probability that the update comes from one of the distributions. The second type of predicates uses the changes of the discrete actuator states to find critical values of sensors. These changes are called events. Each time an event happens in the learning trace, the IDS attempts to predict the value of each sensor with a regression model using the other sensors as features. A correct prediction generates two predicates representing the sensor value before and after the event. After generating the predicates, the IDS uses data mining techniques to find frequent itemsets where the predicates are considered as items. This yields predicates that are often satisfied at the same time. The IDS uses them to generate invariant rules.

Another approach to detect process oriented attacks is the use of a prediction model. Hadžiosmanović et al. [5] present an IDS leveraging autoregression and control limits. The IDS classifies process variables in three categories according to the number of different values it observed in the learning trace. Process variables are constant if they hold only one value, discrete if they hold a few different values, and continuous otherwise. In the learning phase, the IDS stores for constant and discrete process variables all their values. For continuous variables, it creates for each them an autoregressive model that can be used to predict the next value based on p previous values. The order p of the model is automatically estimated. In the detection phase, the IDS performs different kinds of checks. For constant and discrete variables, it looks if values are part of the set of values seen in the learning phase and alerts are raised when it is not the case. For continuous variables, the IDS raises alerts if (i) if the value of a variable is outside of the control limits or (ii) if the value deviates from the prediction model. We used the open source implementation of the autoregressive model described in [5]. However, in its original form it resulted in an *FPR* of 50% and higher on our data. Most of these alerts were raised because of some variables not behaving exactly the same as during the learning phase, even when no attack was performed and also because of the so-called F-test done by the model. We have therefore modified the test.

The open source implementation of the autoregressive model described in [5] has a high FPR when applied to our test data. In the SWaT process, the FPR was higher than 80% and in the simulated process it was close to 50%. Most of these alerts were raised because of some variables not behaving exactly the same as during the learning phase, even when no attack was performed and also because of the F-test. The F-test relies on the size of the two sample sets (residuals and prediction errors) to determine whether the variance of the prediction error is higher, but as the physical process will keep running, the sample size will keep increasing in such a way that the F-test will keep rejecting the null hypothesis (i.e the two variances are the same) even for really insignificant deviations. One solution to that problem would be to limit the sample size but that would add a new operational parameter to the IDS, potentially for each variable. To avoid this additional complexity, we have replaced the F-test by a 6σ test, i.e., we test after each prediction whether the prediction error X is $\mu - 3\sigma \leq X \leq \mu + 3\sigma$ where μ and σ are the mean and standard deviation of the residuals obtained during training. Note that the original assumption of the F-test is that X is normally distributed, therefore $P(\mu - 3\sigma \leq X \leq \mu + 3\sigma)$ should be approximately 99.7% without an attack.

C. CASE STUDY 1: SWaT PHYSICAL PROCESS

1) DESCRIPTION

The SWaT physical process [12] is a scaled down water treatment plant created for cyber-security research and implemented in a testbed with 24 sensors and 27 actuators. The testbed is designed to produce doubly filtered water. Figure 5 gives an overview on the process. It consists of six stages: In the first stage, raw water is moved to a tank and then passed to the second stage where its quality is assessed. In the third, fourth, and fifth stage several undesirable materials are removed. After those stages, the water quality is assessed again and depending on the result the water is either stored or transferred back to the third stage to clean it further.

Adepu and Mathur [12] provide two datasets from the process. The attack-free trace contains the values of the process variables collected over a period of seven days. The attack trace has a duration of four days and contains 36 attacks with different goals, such as attacks that cause an overflow/underflow of tanks, attacks that damage equipment like pipes by controlling the pumps, or wasting resources by overwriting measured values to confuse the control server.

2) RESULTS

Table 4 shows the detection results obtained for I-IDS, the AR-IDS, and our T-IDS. Feng et al. [4] did not make all details of their solution publicly available and we could not reach them, so we tried our best to replicate their results. The first column of Table 4 contains the result they have reported in [4] and the second column gives the results of our re-implementation of their IDS. The differences between the two come from missing parameter values in [4]. We have set those parameters to obtain results as close as possible to the reported results. Despite those differences, we believe that our implementation is close enough to provide a basis for a comparison.

Our T-IDS has a higher DR than I-IDS. As mentioned earlier, invariant-based IDS can, by design, only detect an attack if it breaks an invariant rule. Among the attacks in the attack trace, the majority of them have a negative impact on the physical process that can be detected by I-IDS. However,

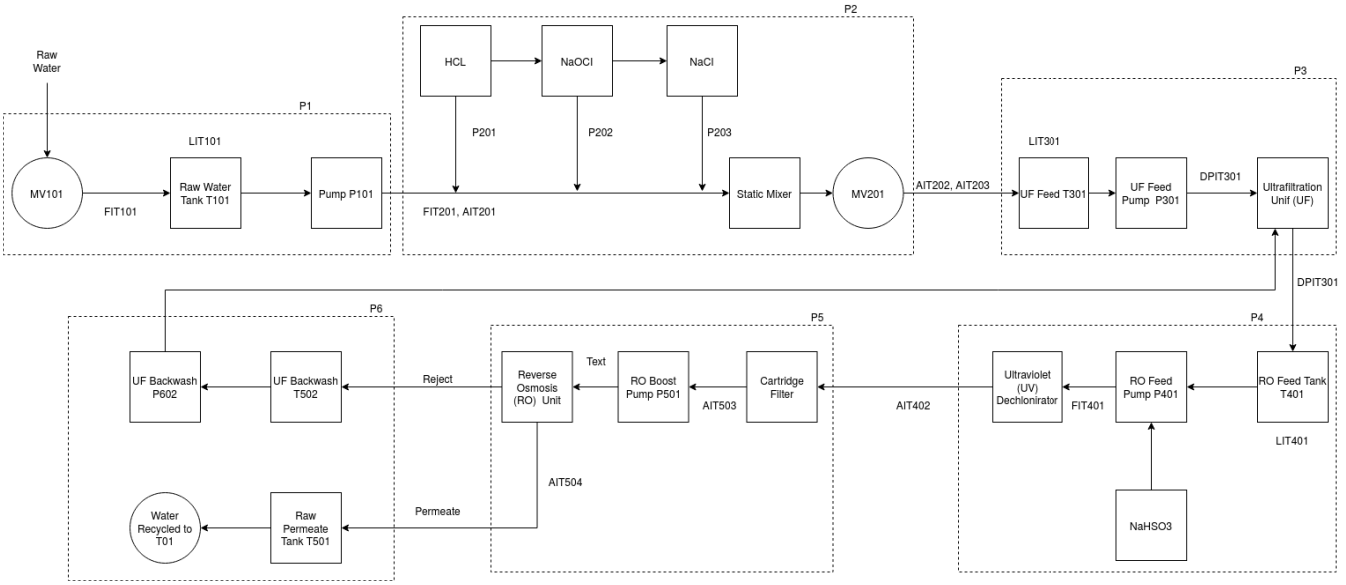


FIGURE 5. SWaT physical process (adapted from [4]).

TABLE 4. Detection results for the SWaT process.

	I-IDS (results from [4])	I-IDS (reimpl.)	AR-IDS (reimpl.)	T-IDS
DR	0.7087	0.6403	0.1747	0.7266
FPR	0.0003	0.0028	0.1424	0.0111

TABLE 5. Confusion matrix: I-IDS.

		Truth Value		
		Positive	Negative	Total
IDS Test	Positive	34474	1279	3573
	Negative	18978	395188	414166
	Total	53452	396467	449919

TABLE 6. Confusion matrix for SWAT: AR-IDS.

		Truth Value		
		Positive	Negative	Total
IDS Test	Positive	10855	55232	66087
	Negative	51248	332584	383832
	Total	62103	387816	449919

TABLE 7. Confusion matrix for SWAT process: T-IDS.

		Truth Value		
		Positive	Negative	Total
IDS Test	Positive	45129	4312	49441
	Negative	16974	383504	400478
	Total	62103	387816	449919

other attacks can be seen more as attempts than successful attacks: Although they measurably influence the physical process, too, the process is able to recover in a few seconds, so no invariant is violated from the viewpoint of I-IDS. This shows the advantage of our T-IDS which is able to detect those attack attempts thanks to the disturbances they cause on the time properties of the physical process. In contrast, AR-IDS is only able to detect a small number of attacks and most of the ones detected are attacks where process variables go beyond their control limits. The main reason for this weak result is the nature of the attacks. AR-IDS is suited to detect sudden changes in process variables but many of the attacks performed on the SWaT process operate slowly and over longer periods. Moreover, some attacks consist of opening or closing valves, but given how discrete variable are treated by AR-IDS, the attacks are ignored because the discrete variables are still in a legitimate state. For the AR-IDS to detect such attacks, continuous variables must be affected as well. These results are shown in more details in table 5, table 6 and table 7 which present the confusion matrix of each IDS on the SWAT process.The x-axis represents the truth value of the data point while the y-axis shows the decision taken by IDS regarding the data-point. A positive (resp. negative) value means that the data-point is malicious (resp. legitimate). As it can be seen in these table, the I-IDS keep the number of false positives low while the T-IDS detects more attack at the expense of raising more alerts, even false alarms.

Detection results per attack type can be found in Table 8. How well an attack is detected by the different IDS depends on its type. Overflow/underflow attacks against tanks are easily detected by all three IDS as they exceed the limits of the variables learned during the training phase. There are also attacks that tamper with the measurements by overwriting the memory of the field devices. Depending on the new values, our T-IDS might detect such attacks if they cause invalid transitions. The AR-IDS detects them because they lead to drastic changes in the stream of values for a variable. I-IDS detects them only if they break an invariant, which is often the case. Finally, some attacks in the attack trace are more subtle. For example, one of them consists in increasing

TABLE 8. Number of detected attacks on SWaT process per type.

	Occurrence	I-IDS	AR-IDS	T-IDS
Actuator modification	12	5	1	7
Actuator blocking	10	4	5	7
Sensor rewriting	25	13	14	17

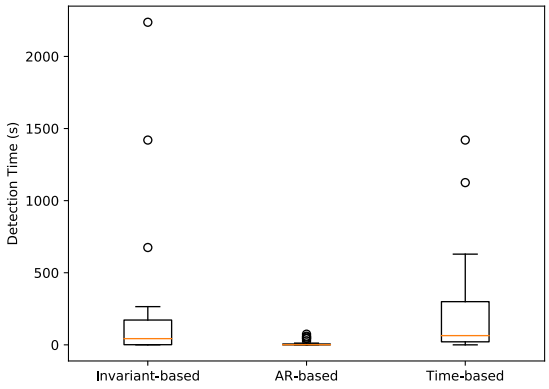


FIGURE 6. Time to detection for the SWaT process.

slightly the value given by a sensor every second. This attack is detected by our T-IDS because it impacts the transition time and the update mean step to the next critical value. It is also detected by I-IDS because the increase of the sensor value does not follow the normal distribution computed for the updates of the sensor. AR-IDS is not able to detect it because the increases are not drastic enough to cause a significant deviation.

3) TIME TO DETECTION

Figure 6 presents the time needed by each IDS to detect attacks. It is computed as the difference between the timestamp of when the first alert concerning an attack was raised and the start of the attack. The average times to detection are 307s (I-IDS), 10s (AR-IDS), and 125s (T-IDS). AR-IDS' low and nearly constant time to detection reflects the fact that it performs a prediction at each update. In contrast, T-IDS has to wait until a critical value is reached or the maximum transition time of the previous critical value is exceeded. I-IDS waits until an invariant is broken. How long this takes depends on the current system state and the type of attack, hence the variability visible in the figure.

4) IMPACT OF PARAMETER δ

Most of the alerts raised by the T-IDS for the SWaT process are the result of abnormally long stillness times of the discrete variables. For example, the stillness times of the two valves MV1 and MV2 are responsible for more than 30% of the alerts. As explained in Section IV, the parameter δ allows to control the detection process performed when the current stillness time exceeds the expected one. Figure 7 shows the impact of different values of δ on the performance of the T-IDS.

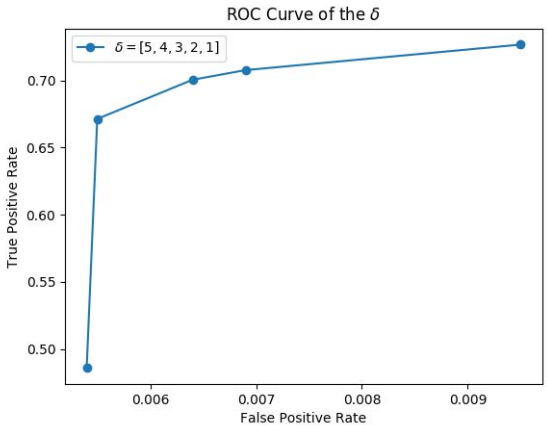


FIGURE 7. Impact of δ on Receiver Operating Characteristics (ROC) for the SWaT process.

D. CASE STUDY 2: SIMULATED PHYSICAL PROCESS

1) DESCRIPTION

In our second experiment, we evaluate the IDS on a simulated physical process. Our starting point is the physical process described in [11] and depicted in Figure 8. Its goal is to release a product by mixing several other products. The process is composed of two stages. In the first stage, 20 units of products are released from *Supp. 1* and *Supp. 2*, in that order. *Motor 1* mixes the products and the result is transferred to *Silo 1*. Then the second stage begins: 20 units of the product stored in *Supp. 3* are carried by a wagon to *Silo 2*. The content of *Silo 1* is added and *Motor 2* mixes them. The final product is then stored in *Tank 2*. The physical process has 23 process variables:

- One for each tank/silo, representing the level of product (7 in total).
- One for each valve representing whether the valve is open or not (7 in total).
- One for each motor representing whether it is running or not (2 in total).
- One for each motor representing the rotating speed of the blades (2 in total)
- Two indicating that the wagon has reached the left resp. right end position.
- One representing whether the wagon is moving.
- One representing the level of product in the wagon.
- One representing whether the trap door of the wagon is opened.

We have modified the process in order to mimic how the behaviour of a physical process could vary in reality, e.g., depending on the consistency of the raw materials. We vary dynamically how fast products are transferred from one container to another. Table 9 gives the flushing times used for the silos and tanks for the physical process, that is, the times to completely empty them, and the filling times, i.e., the times to completely fill them from the supply containers. Note that the supply containers have an infinite capacity and therefore no flushing time. For each flush or fill operation, the time to

TABLE 9. Simulated flushing and filling times.

Container name	Flushing times	Filling times
Tank 1	20s, 4s, 8s	8s, 20s
Silo 1	10s, 8s	–
Silo 2	12s, 6s	–
Wagon	10s, 4s	4s, 10s
Tank 2	12s, 6s	–

complete the operation is picked by the simulator from the table in a round-robin fashion. Furthermore, we increase the speed of *Motor 1* in two steps (from 0 to 20 to 40) when the motor is switched on.

We have implemented the scenario in a simulator written in Python. It simulates both the physical process and the ICS components (MTU and PLCs). The simulation of the process is done with `simpy` [32]. PLC and MTU use `pymodbus` [33] to exchange Modbus/TCP messages on the local interface of the computer. The PLCs and the physical process run in separate python processes and interact through files. The physical process writes the current values of the process variables into a file read by the PLCs. In the opposite direction, the PLCs write to files in order to trigger changes in the physical process, e.g., to open a valve. At the end of the simulation, a dataset is produced as a text file where each line represents the state of the process at a given time. The states are represented as key-value sets where the keys are process variables and the values are the sensors/actuators values.

We generate two traces with the simulator. One attack-free trace of one hour and a trace of 30 minutes where 18 attacks consisting of manipulating sensors and actuators are performed. A complete list of these attacks is provided in Table 10. There are attacks with the goal of disrupting the physical process without completely destroying the process for example by overflowing a tank. Instead, the attacker attempts to slow down the whole physical process by closing valves. The first attack starts ten minutes after the beginning of the simulation by sending commands to the PLCs to close one of the valves for one minute. After the attack, the attacker waits for a period of 5 minutes and then repeats the attack with a different valve and so on. Note that the attacker runs the attacks blindly, i.e., without knowing the current state of the process. Therefore, the impact of an attack and how easily it is detected by the three IDS will depend on the state of the process in the moment when the attack arrives. Other attacks consist of overflow/underflow attacks, and manipulating sensor readings such that the transitions are completed in a valid time even though updates are not done in the expected way.

2) RESULTS

Table 11 shows the detection results. Detection results per attack type can be found in Table 15.

We observe in Table 11 that the simulated process, the I-IDS and the T-IDS roughly have the same number of false positive with a slight advantage to the I-IDS. This can also be observed in Table 12, Table 14. However, the detection rate

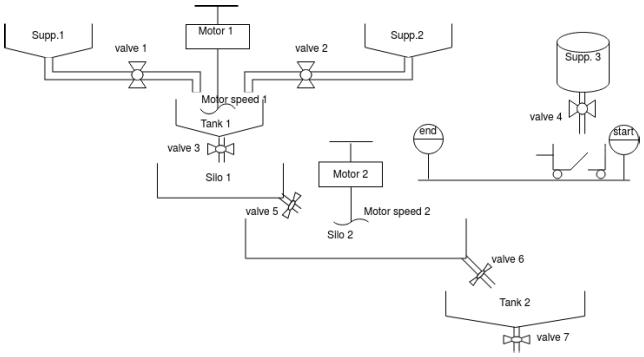


FIGURE 8. The simulated process.

TABLE 10. Attacks on the simulated process and their type.

#	Attacks	Duration
1	Rewriting MotorSpeed1 to reach max speed Attack Type: Sensor rewriting	10s
2	Rewriting MotorSpeed1 to reach max speed Attack Type: Sensor rewriting	10s
3	Closing valve2 and preventing it from opening Attack Type: Actuator modification, Actuator blocking	1m
4	Rewriting MotorSpeed2 to reach max speed Attack Type: Sensor rewriting	10s
5	Rewriting MotorSpeed1 to reach max speed Attack Type: Sensor rewriting	10s
6	Rewriting MotorSpeed1 to reach max speed Attack Type: Sensor rewriting	10s
7	Preventing valve2 from opening Attack Type: Actuator blocking	1m
8	Rewriting MotorSpeed1 to reach max speed Attack Type: Sensor rewriting	10s
10	Rewriting MotorSpeed2 to reach max speed Attack Type: Sensor rewriting	10s
11	Preventing valveTank1 from opening Attack Type: Actuator blocking	1m
12	Rewriting MotorSpeed1 to reach max speed Attack Type: Sensor rewriting	10s
13	Rewriting MotorSpeed1 to reach max speed Attack Type: Sensor rewriting	10s
14	Preventing valve1 from opening Attack Type: Actuator blocking	1m
15	Rewriting MotorSpeed2 to reach max speed Attack Type: Sensor rewriting	10s
16	Rewriting MotorSpeed1 to reach max speed Attack Type: Sensor rewriting	10s
17	Preventing valveTank1 from opening Attack Type: Actuator blocking	1m
18	Rewriting Tank1 to reach max capacity Attack Type: Sensor rewriting	3s

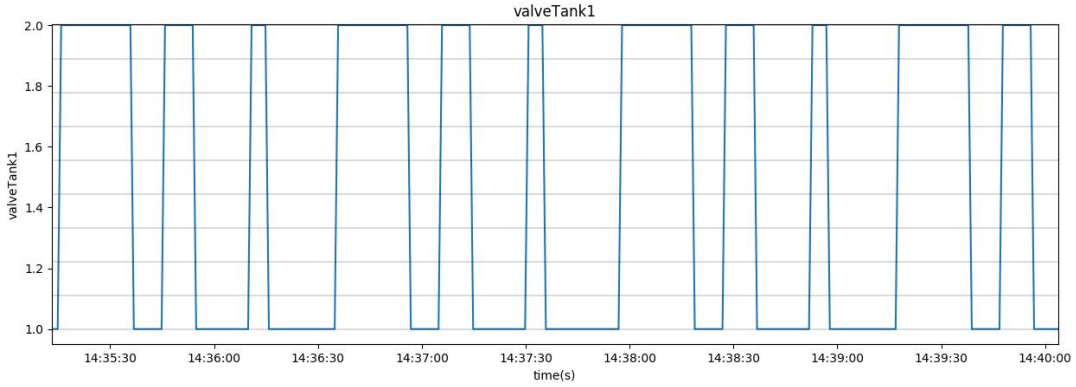
TABLE 11. Detection results for the simulated process.

	I-IDS (reimpl.)	AR-IDS (reimpl.)	T-IDS
DR	0.4541	0.2494	0.7670
FPR	0.0342	0.1149	0.0392

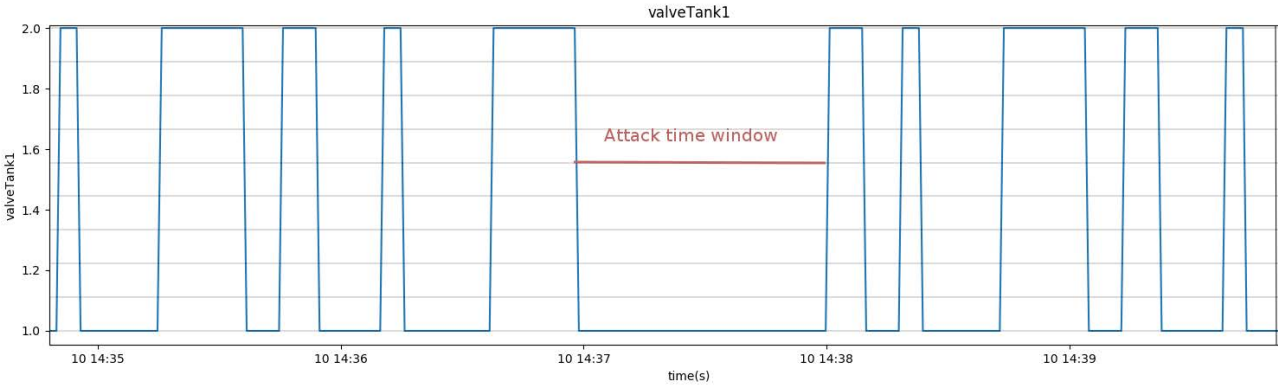
TABLE 12. Confusion matrix for simulated process: I-IDS.

		Truth Value		
		Positive	Negative	Total
IDS Test	Positive	193	48	241
	Negative	232	1353	1585
	Total	425	1401	1826

of the I-IDS is low due to those attacks that were executed in such a way that no invariant was violated. An example



(a) Valve 3 timeseries (normal)



(b) Valve 3 timeseries (attacked)

FIGURE 9. Attack impact on valve 3. (Values on y-axis: 2 ="Open", 1 = "Closed".)

TABLE 13. Confusion matrix for simulated process: AR-IDS.

		Truth Value		
		Positive	Negative	Total
IDS Test	Positive	106	161	267
	Negative	319	1240	1559
	Total	425	1401	1826

TABLE 14. Confusion matrix for simulated process: T-IDS.

		Truth Value		
		Positive	Negative	Total
IDS Test	Positive	326	55	381
	Negative	99	1346	1445
	Total	425	1401	1826

TABLE 15. Detection of attacks on simulated process per type.

	Occurrence	I-IDS	AR-IDS	T-IDS
Actuator modification	1	1	1	1
Actuator blocking	5	3	5	5
Sensor rewriting	12	13	5	13

of such an attack is when the attacker keeps *Valve 3* closed when *Tank 1* has reached 40 units of products. In that case, the normal behavior of the physical process is for *Valve 3* to open and release the product in *Tank 1*. However the state of having 40 units of product in *Tank 1* and the *Valve 3* closed is normal since it is the one occurring just before *Valve 3* is opened by the ICS, therefore no invariant is violated when the attacker keeps that state for one minute. It does not break the process but it disrupts its progress seriously which is undesirable. T-IDS is able to detect the attack as the times of stillness of the process variable of *Valve 3* are impacted. Figure 9 shows how the state of *Valve 3* varies during the normal behavior of the physical process and during an attack. Note, however, that in the case where the attacker closes *Valve 3* when *Tank 1* is getting *empty*, both I-IDS and T-IDS are able to detect the attack: the former because the attack breaks an invariant, and the latter because it impacts the stillness time of *Valve 3* in its "Open" state. Just like for the previous process, the detection rate of AR-IDS is quite low because of how the attacks target the process.

3) TIME TO DETECTION

The detection times are shown in Figure 10. The average times to detection are 2.5s (I-IDS), 2.9s (AR-IDS), and 5.1s (T-IDS). For most of the attacks, T-IDS is faster than the other two IDS. However, for a few attacks, it needs much longer.

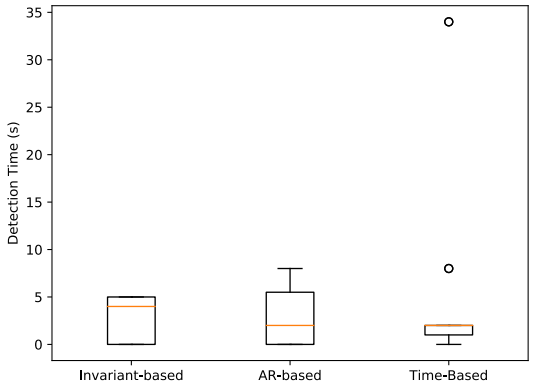


FIGURE 10. Time to detection: Simulated process.

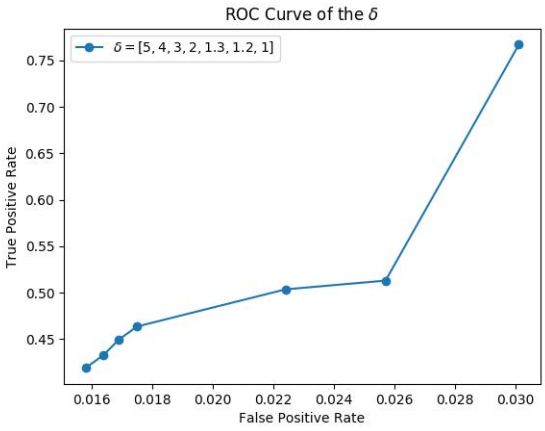


FIGURE 11. Impact of δ on the Receiver Operating Characteristics (ROC) for the simulated process.

This is caused by the way how the attacks are performed. As mentioned earlier, the attacks are run without knowledge of the current state of the process. For example, an attack might try to close a valve even though the valve is already closed. In such a situation, T-IDS has to wait the maximum stillness time of the "Closed" state of the valve before it can raise an alert. In that regard, I-IDS is less susceptible as it captures more explicitly the dependency of that process variable with a large set of invariants that are more likely to be violated early in the case of an anomaly.

4) IMPACT OF PARAMETER δ

For the simulated process, most of the alerts come from sensor transitions, so the value of δ has less impact compared to what was observed with the SWaT process as it can be seen in Figure 11. For values 2, 3, 4, 5, the performance of the T-IDS is quite similar and not very good. This can be explained by the fact that blocking attacks are missed because the T-IDS is waiting too long to raise an alert. Therefore, by decreasing δ, the *TPR* increases. To conclude, the default value of 1 seems to be a good trade-off for both processes. It should be noted that the number of data points for this process is smaller than in the SWaT process, so missing a malicious point has a higher impact on the computation of the *TPR* (same for *FPR*).

E. EXECUTION TIME

All three tested IDS are able to process the data faster than real-time in the detection phase. For example, they need less than 25 seconds to classify one hour of data of the simulated process. For the learning phase, T-IDS and AR-IDS need around 30 minutes to process the entire training data, while I-IDS needs several hours due to the complexity of its data mining process.

VI. DISCUSSION AND LIMITATIONS

A. CHANGES IN THE PHYSICAL PROCESS OR EQUIPMENT

In our experiment the physical process behavior is static, in the sense that it stays the same over time. There could be several reasons why a process changes its behavior over time. If there is an intentional change (for example, the plant operator decides to permanently change the temperature at which a chemical reaction should run), the detection model has to be retrained. If the change is unintentional, e.g., due to the aging of the physical equipment, the impact on the IDS depends on the nature of the change and how the ICS operator handles it. For example, if the aging of a sensor results in more noise or in a bias, it can happen that the measurement values provided by that sensor will at some point exceed the tolerance learned during the learning phase. If the operator considers the inaccurate sensor values as acceptable, an adaptation of the detection model will be necessary to reduce the number of false alerts.

On the other hand, it is in the operator's own interest to have correct sensor readings. Many PLCs allow to configure how an analog sensor input is filtered and converted to a digital value. If the operator uses this feature to correct the sensor reading directly on the PLC, the IDS will not need to be retrained since it is getting its data from the server logs and not directly from the physical equipment.

B. LIMITATIONS OF THE DETECTION

Our time-based IDS focuses on recurrent variables with critical values. This means that if an attack is done on non-recurrent variables, it might not be detected. However, given all the dependencies that exist between variables in physical processes, an attack against one process variable has a high chance to impact others as well, meaning that the IDS might not necessarily pinpoint which process variable was targeted by an attack, but it can still detect the incident. This is what we observed in the SWaT process for the attacks number 6, 8, and 10 in the original dataset (please refer to [12] and [34] for a complete list of attacks). Attack 6 rewrites the values of an acidity sensor in stage 2 of the physical process (see Figure 5). The IDS detects the attack due to its impact it has two hours later on the values measured by another acidity sensor in stage 5 of the process. Attacks 8 and 10 overwrite the values of pressure and flow sensors in stages 3 and 4, respectively, and

are quickly detected by the IDS because the changes caused by the attacks in the treated water affect the behavior of a water level sensor and pump in neighbor stages.

In order for an attacker to bypass the IDS, they would need to preserve the behavior of the targeted variables to some extent. As illustrated above, achieving such a feat is extremely difficult for complex processes and would require a detailed knowledge of all the dependencies of the processes and their variables, even the ones that are implicit and are not directly shown in the design of the process. Moreover, the dependencies between the variables force an attacker to be more active as they need to impact several variables at the same time which is technically costly. Another way an attacker could try to bypass the system could be to carefully and slowly tamper with the updates of the sensors by adding or subtracting small values. However, this would impact the transition time and the mean update step and therefore would likely be detected.

Another limitation is the time to detection. In order to detect an attack, our IDS has to wait until a transition happens or until the longest possible transition time or stillness time for the last reached critical value is exceeded. For this reason, the IDS is not able to identify all the malicious snapshots of the physical process in the second case study, and the time it will take for the IDS to detect an attack will depend on the timing of the attacker. Imagine a situation where a valve has just been opened. If the attacker immediately closes the valve, the IDS notices the transition from critical state ''Open'' to ''Closed''. However, if the valve was already closed and the attacker prevents it from opening, the IDS has to wait the maximum stillness time of the valve's ''Closed'' state.

Our IDS does not, in its current form, analyze exhaustively the correlation between variables. This is a strength of the invariant-based approach [4]. To a certain degree, some level of correlation is implicitly taken into account as the dependencies between the variables are used to identify the critical values. Nevertheless, in view of the low detection rate of the invariant-based approach in the second test case, we observe that our IDS does remarkably well.

The last point we would like to address is the question of attack detection versus attack classification. Currently, the IDS only raises an alert in the presence of an attack and only gives information about which transitions is considered malicious and why. The IDS does not provide direct human-readable information on the *type* of attack. For example, the IDS does not distinguish between an attack that slightly modifies sensor values and one that blocks an actuator. This kind of information could be useful for security officers to understand how an attack is performed and what action should be taken as a response. To our knowledge, there exists in the literature a taxonomy of attacks on Industrial Control Systems, but only at the network and software level [2]. There is currently no taxonomy for process-oriented attacks. We leave the creation of such a taxonomy and the definition of attack classes to future work.

VII. CONCLUSION

Intrusion Detection Systems (IDS) that have been designed for the protection of general-purpose ICT networks can also be used to protect industrial control systems (ICS). However, they do not understand the physical process monitored by the ICS. This lack of process awareness allows attackers with knowledge of the process semantics to perform attacks harming the process itself. *Process aware* IDS are designed to detect such attacks.

In this paper, we look at learning-based process aware IDS, i.e., process aware IDS that learn the expected behavior of the ICS and the underlying physical process from past observations. Existing approaches proposed in the literature based on the learning of invariants tend to result in very few false positives for process-oriented attacks. However, some types of attacks, e.g., attacks that aim at slowing down the physical process, evade detection because they are designed to not violate invariants.

We propose a time-based process aware IDS that is able to learn the temporal properties of the physical process from a series of snapshots of the process variables. It is fast and only depends on a few configuration parameters. We have evaluated our solution on data traces from a real SCADA testbed and a simulated ICS that contain various types of attacks. Comparisons with an invariant-based IDS and a prediction-based IDS taken from the literature show that our IDS achieves a significantly higher detection rate.

As future work, we consider models that can also express time-based relationships between different process variables.

The code and datasets can be found at: https://github.com/gkabasele/ProcessBasedIDS.git

REFERENCES

[1] K. A. Stouffer, ''SP 800–82. Guide to industrial control systems (ICS) security: Supervisory control and data acquisition (SCADA) systems, distributed control systems (DCS), and other control system configurations such as programmable logic controllers (PLC),'' Gaithersburg, MD, USA, Special Publication 800-82, 2011.

[2] B. Zhu, A. Joseph, and S. Sastry, ''A taxonomy of cyber attacks on SCADA systems,'' in *Proc. Int. Conf. Internet Things 4th Int. Conf. Cyber, Phys. Social Comput.*, Oct. 2011, pp. 380–388.

[3] N. Goldenberg and A. Wool, ''Accurate modeling of modbus/TCP for intrusion detection in SCADA systems,'' *Int. J. Crit. Infrastruct. Protection*, vol. 6, no. 2, pp. 63–75, Jun. 2013.

[4] C. Feng, V. R. Palleti, A. Mathur, and D. Chana, ''A systematic framework to generate invariants for anomaly detection in industrial control systems,'' in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2019, pp. 1–22.

[5] D. Hadžiosmanović, R. Sommer, E. Zambon, and P. H. Hartel, ''Through the eye of the PLC: Semantic security monitoring for industrial processes,'' in *Proc. 30th Annu. Comput. Secur. Appl. Conf.*, 2014, pp. 126–135.

[6] J. Slay and M. Miller, ''Lessons learned from the Maroochy water breach,'' in *Critical Infrastructure Protection*, E. Goetz and S. Shenoi, Eds. Boston, MA, USA: Springer, 2008, pp. 73–82.

[7] R. Flosbach, J. J. Chromik, and A. Remke, ''Architecture and prototype implementation for process-aware intrusion detection in electrical grids,'' in *Proc. 38th Symp. Reliable Distrib. Syst. (SRDS)*, Oct. 2019, pp. 42–4209.

[8] H. Lin, A. Slagell, Z. Kalbarczyk, P. W. Sauer, and R. K. Iyer, ''Runtime semantic security analysis to detect and mitigate control-related attacks in power grids,'' *IEEE Trans. Smart Grid*, vol. 9, no. 1, pp. 163–178, Jan. 2016.

[9] G. Constante, C. Moya, and J. Wang, "Semantic-based detection architectures against monitoring-control attacks in power grids," in *Proc. IEEE Int. Conf. Commun., Control, Comput. Technol. Smart Grids (SmartGridComm)*, Oct. 2019, pp. 1–6.
[10] D. I. Urbina, J. A. Giraldo, A. A. Cardenas, N. O. Tippenhauer, J. Valente, M. Faisal, J. Ruths, R. Candell, and H. Sandberg, "Limiting the impact of stealthy attacks on industrial control systems," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2016, pp. 1092–1105.
[11] O. Koucham, S. Mocanu, G. Hiet, J.-M. Thiriet, and F. Majorczyk, "Detecting process-aware attacks in sequential control systems," in *Proc. 21st Nordic Conf. Secure IT Syst. (NordSec)*, Oulu, Finland, Nov. 2016, pp. 20–36. [Online]. Available: https://hal.inria.fr/hal-01361081
[12] S. Adepu and A. Mathur, "Using process invariants to detect cyber attacks on a water treatment system," in *Proc. IFIP Int. Conf. ICT Syst. Secur. Privacy Protection*. Midtown Manhattan, NY, USA: Springer, 2016, pp. 91–104.
[13] Modbus. (2006). *Modbus Messaging on TCP/IP Implementation Guide V1.0b*. [Online]. Available: http://www.modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf
[14] V. M. Igure, S. A. Laughter, and R. D. Williams, "Security issues in SCADA networks," *Comput. Secur.*, vol. 25, no. 7, pp. 498–506, Oct. 2006.
[15] S. Selvarajan, M. Shaik, S. Ameerjohn, and S. Kannan, "Mining of intrusion attack in SCADA network using clustering and genetically seeded flora-based optimal classification algorithm," *IET Inf. Secur.*, vol. 14, no. 1, pp. 1–11, 2020.
[16] W. Gao, T. Morris, B. Reaves, and D. Richey, "On SCADA control system command and response injection and intrusion detection," in *Proc. eCrime Researchers Summit*, 2010, pp. 1–9.
[17] M. Caselli, E. Zambon, and F. Kargl, "Sequence-aware intrusion detection in industrial control systems," in *Proc. 1st ACM Workshop Cyber-Phys. Syst. Secur.*, New York, NY, USA, Apr. 2015, pp. 13–24, doi: 10.1145/2732198.2732200.
[18] R. R. R. Barbosa, R. Sadre, and A. Pras, "Exploiting traffic periodicity in industrial control networks," *Int. J. Crit. Infrastruct. Protection*, vol. 13, pp. 52–62, Jun. 2016.
[19] S. Adepu and A. Mathur, "From design to invariants: Detecting attacks on cyber physical systems," in *Proc. IEEE Int. Conf. Softw. Qual., Rel. Secur. Companion (QRS-C)*, Jul. 2017, pp. 533–540.
[20] V. Menzel, J. L. Hurink, and A. Remke, "Securing SCADA networks for smart grids via a distributed evaluation of local sensor data," in *Proc. IEEE Int. Conf. Commun., Control, Comput. Technol. Smart Grids (SmartGridComm)*, Oct. 2021, pp. 405–411.
[21] M.-H. Monzer, K. Beydoun, A. Ghaith, and J.-M. Flaus, "Model-based IDS design for ICSs," *Rel. Eng. Syst. Saf.*, vol. 225, Sep. 2022, Art. no. 108571.
[22] F. Sicard, E. Zamai, and J.-M. Flaus, "Critical states distance filter based approach for detection and blockage of cyberattacks in industrial control systems," in *Diagnosability, Security and Safety of Hybrid Dynamic and Cyber-Physical Systems*, M. Sayed-Mouchaweh, Ed. Cham, Switzerland: Springer, 2018, pp. 117–145.
[23] A. Carcano, A. Coletta, M. Guglielmi, M. Masera, I. N. Fovino, and A. Trombetta, "A multidimensional critical state analysis for detecting intrusions in SCADA systems," *IEEE Trans. Ind. Informat.*, vol. 7, no. 2, pp. 179–186, May 2011.
[24] C.-Y. Lin, S. Nadjm-Tehrani, and M. Asplund, "Timing-based anomaly detection in SCADA networks," in *Proc. Int. Conf. Crit. Inf. Infrastruct. Secur.* Midtown Manhattan, NY, USA: Springer, 2017, pp. 48–59.
[25] C.-Y. Lin and S. Nadjm-Tehrani, "Timing patterns and correlations in spontaneous SCADA traffic for anomaly detection," in *Proc. 22nd Int. Symp. Res. Attacks, Intrusions Defenses (RAID)*, 2019, pp. 73–88.
[26] V. K. Singh, H. Ebrahem, and M. Govindarasu, "Security evaluation of two intrusion detection systems in smart grid SCADA environment," in *Proc. North Amer. Power Symp. (NAPS)*, Sep. 2018, pp. 1–6.
[27] Z. Yang, L. He, P. Cheng, J. Chen, D. K. Yau, and L. Du, "PLC-sleuth: Detecting and localizing PLC intrusions using control invariants," in *Proc. 23rd Int. Symp. Res. Attacks, Intrusions Defenses (RAID)*, 2020, pp. 333–348.
[28] H. R. Ghaeini, D. Antonioli, F. Brasser, A.-R. Sadeghi, and N. O. Tippenhauer, "State-aware anomaly detection for industrial control systems," in *Proc. 33rd Annu. ACM Symp. Appl. Comput.*, Apr. 2018, pp. 1620–1628.
[29] Y. Cheng, "Mean shift, mode seeking, and clustering," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 17, no. 8, pp. 790–799, Aug. 1995.
[30] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "LOF: Identifying density-based local outliers," *SIGMOD Rec.*, vol. 29, no. 2, p. 93–104, May 2000, doi: 10.1145/335191.335388.
[31] G. F. Jenks, "The data model concept in statistical mapping," in *International Yearbook of Cartography*, vol. 7, 1967, pp. 186–190.
[32] *Discrete Event Simulation for Python*. [Online]. Available: https://bitbucket.org/simpy/simpy/src/default/
[33] Pymodbus. (2000). *A Full Modbus Protocol Written in Python*. [Online]. Available: https://github.com/riptideio/pymodbus
[34] S. U. o. T. iTrust, Centre for Research in Cyber Security and Design. (2018). *Secure Water Treatment (SWAT)*. [Online]. Available: https://itrust.sutd.edu.sg/itrust-labs_datasets/dataset_info/

GORBY KABASELE NDONDA received the Ph.D. degree, in March 2022. He is currently a Postdoctoral Researcher at the Université Catholique de Louvain (UCLouvain). His research interests include security of industrial control systems, detection of intrusions, and software-defined networking.

RAMIN SADRE (Member, IEEE) was an Assistant Professor at Aalborg University and a Postdoctoral Researcher at the University of Twente. He is currently a Professor of security and performance of networked systems at the Université Catholique de Louvain (UCLouvain). His research interests include security of the Internet of Things and industrial control systems, in particular the detection of intrusions.