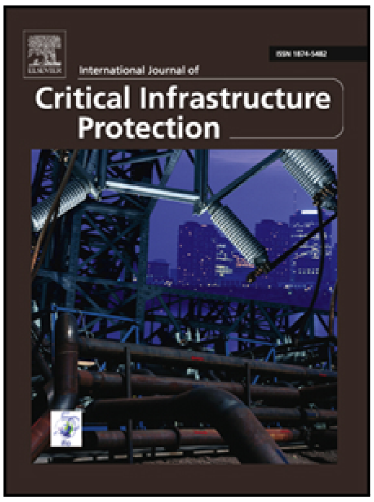


On the practical integration of anomaly detection techniques in industrial control applications

Piroska Haller, Béla Genge, Adrian-Vasile Duka

PII: S1874-5482(18)30102-1
DOI: https://doi.org/10.1016/j.ijcip.2018.10.008
Reference: IJCIP 277

To appear in: *International Journal of Critical Infrastructure Protection*

Received date: 10 August 2018
Revised date: 5 October 2018
Accepted date: 22 October 2018

On the practical integration of anomaly detection techniques in industrial control applications

Piroska Haller, Béla Genge, Adrian-Vasile Duka

Department of Computer Science, Petru Maior University of Tirgu Mures, Mures, Romania, 540088

Abstract

Despite significant advances made on anomaly detection systems, few reports are found documenting their practical integration into the industrial realm. Furthermore, the literature reports a wide range of complex detection strategies, which may require hardware changes/updates in order to be supported by critical industrial equipment such as industrial controllers (e.g., Programmable Logic Controllers). To address these issues, this paper documents a systematic methodology for the practical integration of lightweight anomaly detection algorithms into industrial control applications. It shows that industrial controllers, and in particular the scheduling rate of user programs, are sensitive to network traffic-based disturbances. Therefore, the methodology embraces the task scheduling rates found in control applications, and their deviation from the "normal" behavior. It designs a "monitoring" task, and an innovative algorithm for detecting abnormal task scheduling rates by leveraging the cumulative sum model (CUSUM) and a regression strategy applied on a specific time interval. Essentially, the approach enhances the industrial controller with a "security module" that can trigger alerts to identify early cyber attacks. The approach is extensively analyzed in the context of two industrial controllers: a Phoenix Contact ILC 350-PN controller, and a Siemens SIMATIC S7-1200 Programmable controller.

Email addresses: `phaller@upm.ro` (Piroska Haller), `bela.genge@ing.upm.ro` (Béla Genge), `adrian.duka@ing.upm.ro` (Adrian-Vasile Duka)

Keywords: Anomaly Detection, Industrial Control System, Programmable Logic Controller, cumulative sum.

1. Introduction

Industrial controllers (e.g., Programmable Logic Controllers, Remote Terminal Units) are, in most cases, embedded devices specialized for real-time applications in manufacturing and process control. These are available in a wide variety of configurations, running a diverse palette of operating systems together with dedicated real-time schedulers. While these controllers are designed to deliver robust and effective control strategies, little has been done towards the integration of security solutions within their application layer. In fact, in the vast majority of cases, control applications do not account for the security and the inner monitoring of their behavior. This immense responsibility has been transferred to external devices such as "bump-in-the-wire" monitoring devices (Intrusion/Anomaly Detection Systems, process monitoring systems), and cryptographic devices providing the secure tunneling of legacy industrial protocols (e.g., IPSec).

To this end, a considerable amount of research has been focused on the development of Intrusion Detection Systems (IDS) [1, 2, 3, 4, 5] for the industrial realm. Different strategies have been developed by leveraging diverse techniques such as classification [6, 7, 8], multivariate statistical analysis including principal component analysis [9, 10], and data fusion [11, 12, 13]. Nevertheless, we observe that the practical implementation of previous methodologies within the industrial realm would require major software/hardware changes. Furthermore, in most cases, the complexity of the suggested detection strategy does not permit their integration within the application running on industrial controllers. This is owed to the time constraints imposed to the scheduling of real-time control applications, where complex computations may significantly affect the schedulability of such applications.

Based on the aforementioned issues, this paper presents a methodology for integrating anomaly detection systems into control applications. At its core, the methodology embraces the task scheduling rates found in control applications, and their deviation from the "normal" behavior. It proposes an innovative methodology for detecting the abnormal behavior by leveraging the cumulative sum model (CUSUM) and a regression strategy applied on a specific time interval. Furthermore, a simplistic implementation is presented for reducing its complexity and the memory requirement. The approach demonstrates that the continuous observation of the scheduling rate of control applications from a dedicated "monitoring" task constitutes a significant metric for the detection of external disturbances. Accordingly, we show that industrial controllers (IC) are sensitive to network discovery attacks, as well as to Denial of Service (DoS) attacks. The former case constitutes a significant step in the construction of a cyber kill chain [14, 15], a technique adopted from the military domain to denote the steps followed in the deployment of a targeted cyber attack. We note that, while previous reports have confirmed the sensitivity of IC to network discovery attempts [16], the developed methodology provides an effective detection strategy that can signal network discovery attempts and DoS attacks at their early stage. Accordingly, the approach can be integrated into early warning systems [17] to issue alarms and prevent possibly damaging attacks.

Extensive experimental investigations are conducted in the context of two IC: a Phoenix Contact ILC 350-PN, and a Siemens SIMATIC S7-1200 Programmable controller. The experiments show that IC are indeed sensitive to network discovery and DoS attacks. Furthermore, they prove that the proposed implementation can be readily integrated into a wide variety of IC. Throughout this paper we make the following main contributions:

- We formulate a methodology for integrating anomaly detection systems within the application of industrial controllers.
- We propose a simple and efficient detection engine based on the cumulative sum model and a regression strategy.
- We develop a simplistic implementation of the proposed methodology that

can be readily integrated into IC.

- We present extensive experimental results on two well-established IC: a Phoenix Contact ILC 350-PN and a Siemens SIMATIC S7-1200 Programmable controller, which represent two widely used IC found in the automation systems of Romanian natural gas transportation networks.

The remainder of this paper is organized as follows. Section 2 provides an introduction to the architecture of Industrial Control Systems, and an overview of related studies. Section 3 describes the proposed methodology for detecting abnormal task scheduling rates, while Section 4.2 documents the implementation of the proposed detection algorithm. Experimental results based on ILC 350-PN controllers are presented in Section 5, and the results concerning the Siemens SIMATIC S7-1200 Programmable controller are detailed in Section 6. The paper concludes in Section 7.

2. Background and related work

2.1. Overview of industrial control systems

The architecture of modern Industrial Control Systems (ICS) can be viewed as a unique technological ecosystem consisting of various devices ranging from sensors and actuators, industrial equipment, video surveillance cameras, to traditional personal computers and networking devices. In terms of communications we find a broad range of technologies (traditional and industry-grade) including wired and wireless. At their core, ICS include the Supervisory Control And Data Acquisition (SCADA) system, which embraces all the Information and Communication Technologies (ICT) hardware and software required to monitor the physical process and to implement control loops. Here, we find industrial controllers (e.g., Programmable Logic Controllers - PLCs), and Remote Terminal Units (RTUs) that read data from sensors and produce the local control strategy by issuing control signals to actuators. These controllers also handle the communication requirements of the industrial system, by exchanging data

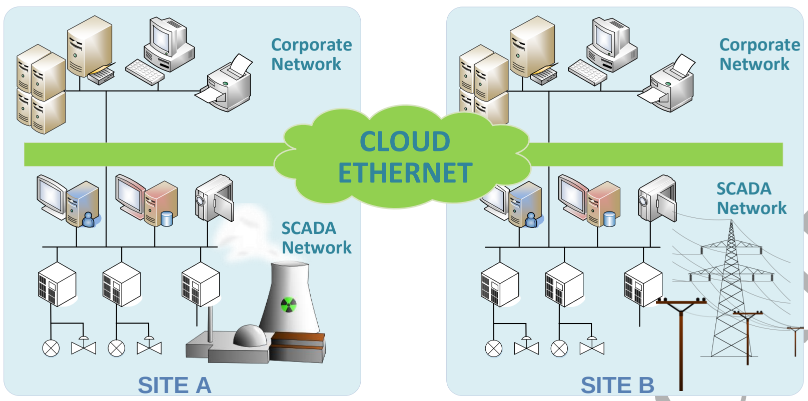


Figure 1: Architectural overview of modern ICS.

with other field devices, as well as, with the SCADA servers, and, eventually executing the received commands.

The communications between SCADA servers and PLCs is usually implemented in two ways: (i) through an Open Platform Connectivity (OPC) layer that helps map the PLC devices, program, and monitor the hardware controllers; and/or (ii) through a direct memory mapping notation, which makes use of communications protocols such as Modbus and DNP3.

The architecture of ICS can be distributed across large geographical regions interconnected by state of the art Cloud-based solutions. To this end, there have been many recent developments and applications based on cloud-assisted industrial systems [18]. These enable flexible processing of vast amounts of data, they can provide on-demand services and applications, with built-in support for redundancy and data availability. An example architecture of an ICS is shown in Figure 1.

2.2. Related work

The dramatic impact of traditional computer system attacks on industrial processes has been first demonstrated by the Stuxnet malware [19, 20]. In light of such threats, there have been numerous proposals towards enhancing the security of existing and future industrial installations [21, 22]. In fact, a considerable amount of research has been allocated to understanding the design of comprehensive anomaly detection systems for the industrial realm [23]. In the remaining of this section we provide an overview of the main anomaly detection techniques for ICS available in the scientific literature. We observe that methodologies can be categorized according to the scope of the monitored parameters. Accordingly, we can distinguish between: (i) process-specific detection methods, where the features exposed by the industrial processes constitute the input for the detection strategy; (ii) network intrusion detection systems, where the industrial traffic is closely monitored and used to signal an abnormal behavior; and (iii) hybrid detection techniques combining process-specific parameters with network intrusion detection techniques.

2.2.1. Process-specific detection strategies

In terms of process-specific detection methodologies we start with the work of Cárdenas, *et al.* [24], which demonstrated that, by incorporating knowledge on the physical process in the control loop, it is possible to detect traditional computer attacks that change the behavior of the underlying physical process. Consequently, by leveraging the knowledge of the physical process, the detection can focus on the attack target, rather than on the various strategies that an adversary may undertake. More recently, in [25], Giraldo, *et al.*, stepped further and designed an attack-resilient controller. In terms of detection, [25] adopted the non-parametric cumulative sum model.

The work of Nai Fovino, *et al.* [1, 26] builds on the assumption that every attack on ICS will ultimately lead to a transition of the system from a *secure state* to a *critical state*. Critical state descriptions are created by process engineers and are used by detection engines to estimate the state of the process. Essentially, in their work, the authors elaborate a new critical state-based industrial firewall that blocks commands that would force the process to a critical state.

More recently, the applicability of data clustering techniques for anomaly detection was explored by the work of Kiss, *et al.* [6]. The Gaussian mixture model was compared to the K-means clustering technique, and the superior performance of the former was demonstrated in the context of a chemical process. A similar attempt for the classification of different events was undertaken by Wang and Mao in [8]. Here, an ensemble of two models of one-class classification was developed. Its performance was demonstrated in the context of two industrial installations (an electric arc furnace and a wind tunnel) and several public datasets. In the direction of multivariate statistical analysis we find the work of Daegeun Ha, *et al.* [9]. Here, the multi-mode principal component analysis (PCA) was used together with the K-nearest neighbor algorithm for process monitoring and data classification. The approach was evaluated in the context of a mixed refrigeration physical process. In a similar direction, Portnoy, *et al.* [10] developed a weighted adaptive recursive PCA approach for fault detection in a natural gas transmission pipeline. Conversely, Chen, *et al.* [13], aimed at reducing the size of the monitored parameter space with the help of a multisensor fusion strategy. The approach was tested in the context of state estimation of a small power network.

Especially in power systems, the problem of defending against cyber attacks received significant attention. Accordingly, Zhao, *et al.* [27], used the measurements from a limited number of Phasor Measurement Units (PMUs) in conjunction with a robust projection statistics to perform effective statistical consistency verification against measurements. In [28], the secure state estimation problem was formulated as a non-convex minimization problem. In the same work, the satisfiability modulo theory was used to derive a detection engine for the power system's abnormal behavior.

2.2.2. Network traffic-based detection strategies

In terms of network-specific methodologies we find the work of Pleijsier [29], where it was shown that by inspecting connection parameter patterns between client and server stations, abnormal communication behavior can be detected. In a similar way Barbosa, *et al.* [30, 31, 32] and Garitano, *et al.* [33], proved that the periodicity of network traffic can be the basis for detecting anomalies in the cyber dimension of ICS. In the same direction, the work of Genge, *et al.* [34] used network traffic flow white-listing in the construction of a Snort-based detection engine for ICS communications.

More recently, the work of Bernieri, *et al.* [35], documented the steps for the detection of network traffic anomalies in critical industrial systems. Their detection engine comprises a network analyzer component, a network profile generator, and an anomaly detection engine that issues an alert if a certain threshold is exceeded for a specific monitored parameter. Cheminod, *et al.* [36] developed a methodology that can automatically compute the sequence of vulnerabilities that can be exploited by an attacker. The methodology leverages the system description and vulnerability databases such as Common Vulnerabilities and Exposures (CVE) [37].

The recent work of Markman, *et al.* [38], however, contradicts the previous work on network traffic patterns. According to their study, instead of a cyclic Deterministic Finite Automaton (DFA), a burst-DFA model would fit the data much better.

2.2.3. Hybrid detection strategies

The combination of both network and process-specific detection methodologies also received significant attention. The work of Genge, *et al.* [11], explored the fusion of parameters from the cyber and the physical realms in the attempt to combine evidence in a multi-dimensional detection strategy. In a similar direction, Di Pietro, *et al.* [12], developed a situational awareness methodology by leveraging a distributed data fusion methodology and combining evidence from a wide variety of sensors. In [39], Caselli, *et al.* explored the semantic of permitted events in the development of an intrusion detection system. Furthermore, the approach identified specific event sequences that formed the basis for the definition of a discrete-time Markov chain that encompasses both network and process-specific parameters. The work of Wan, *et al.* [7], integrated the monitoring of network characteristics with the process-specific parameters. In their work, a weighted mixed Kernel function and parameter optimization methods were developed to improve the performance of the classification.

2.2.4. Discussion

According to this analysis we emphasize that, while a wide variety of techniques have been proposed, few cases addressed the practical integration of detection strategies within the industrial realm. Therefore, given the complexity of the detection schemes such as the ones proposed in [6, 7, 8, 9, 10, 11, 12, 13], their implementation in IC applications may require external hardware. Conversely, the integration of process-specific information in the design of the control system, as proposed in [24] and [25], may represent a feasible solution. However, the authors do not provide specific details on the practical implementation of such control strategies. Furthermore, their proposal focuses on the monitoring of the physical process, while in the paper at hand, our aim is to monitor the operation of the control hardware from the application layer, and to detect abnormal behavior accordingly.

A notable aspect of the methodology proposed in this work is that it showcases the feasible integration of detection strategies within two widely used IC. The work portrays a significant progress towards the practical provisioning of anomaly detection systems, and a paradigm shift in terms of their positioning (i.e., at the application layer of control applications). Consequently, it advocates that security mechanisms can be properly tailored for being accommodated into IC, in spite of their modest computational features, when compared to traditional computer systems. As a result, each controller can be enhanced with a basic "security module" that can trigger alerts to identify early cyber attacks.

3. Proposed methodology

3.1. Architecture of IC

The architecture of modern IC can vary according to different vendors. Accordingly, on top of the hardware we may find a classical operating system (e.g., Linux Version 2.6 or later, FreeRTOS OPENRTOS, RTX, Windows CE

6.0, Windows Embedded Compact 7, etc.) together with a dedicated real-time operating system (e.g., ProconOS) or a real-time scheduler [40]. Irrespective on the underlying solution, control code is usually organized in several distinct *user* tasks, which are scheduled for periodic execution by the real-time preemptive scheduler (or the real-time operating system) governing the temporal determinism of tasks.

Besides user tasks, IC also host *system* tasks that implement the typical functions for handling remote connections, processing of network packets, reading sensor values, and self-diagnosis. The scheduler uses the task priority levels in the task scheduling algorithm. To this end, the processor time is always assigned in the favor of user tasks in order to ensure that critical control functions are executed in the expected deadline. Consequently, in the case of increased load (e.g., due to application state changes, or network traffic-based attacks), the tasks that are in charge of communications (e.g., via Modbus/TCP, OPC) are among the first to exhibit a change in their scheduling behavior. This is owed to the fact that, underneath, communications are handled by interrupt service routines (ISR) that, in case of disturbances (e.g., increasing number of packets), are overloaded with processing requests. As a result, the execution of the ISR will require additional computational resources, which delay the scheduling of user tasks.

User tasks can be classified as: cyclic tasks, event tasks, and default tasks. Cyclic tasks are activated periodically (i.e., they change their state from "ready to run"), but they are actually scheduled to run according to their configured priority and deadline. The default task has the lowest priority and is executed when no other task is active. Conversely, event-based tasks are activated by the OS when the ISR finishes its execution. The actual scheduling time of the event task also depends on the priority of the task.

The time between two consecutive runs of the same periodic task (difference between the start time) might exhibit a difference from the task period even for the highest priority task. This can be owed to a large number of hardware interrupts, and to the execution time of the interrupt service routines. The start

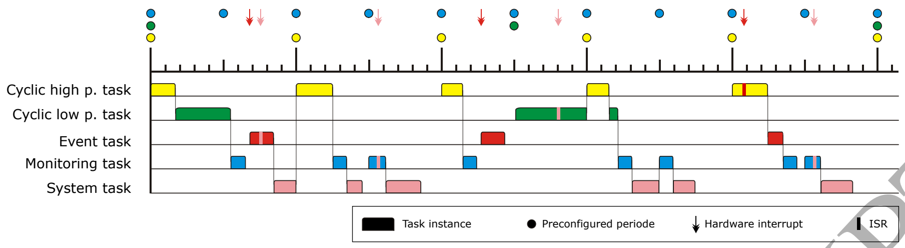


Figure 2: Example task scheduling, which includes the proposed *monitoring* task.

and end time of low priority cyclic tasks are difficult to predict as they depend on the execution time of high priority tasks, the number of interrupts, and the overhead in the OS kernel. However, the finishing time for a task should be less than a predefined watchdog time, otherwise an error event is generated at run-time.

An example task scheduling scenario is depicted in Figure 2. Here, we depicted both cyclic and event-based tasks, together with the monitoring task, and system tasks. The monitoring task is a salient feature of the proposed methodology. It continuously records its own scheduling rate, and runs the proposed detection algorithm. As shown in this figure, high priority tasks are scheduled with a higher accuracy, while the execution of low priority tasks is interrupted and delayed by the high priority tasks. We further observe that task execution is also interrupted by ISRs, which are called as a response to external events. A particularly interesting aspect is that, an increasing number of executed ISRs can delay the planning of user tasks (even high priority tasks). Therefore, the careful monitoring of task scheduling delays can yield an effective instrument in the design of an anomaly detection system (ADS) for IC.

In terms of communications via memory mapping-based protocols such as Modbus, IC host a dedicated user task that periodically checks the status of the Modbus connection, and it reads/writes the Modbus data packet to/from the registers. We observe that when the Modbus packets are processed, the execution time of the Modbus task also increases, whereas the worst case execution time can be precisely computed according to the maximum Modbus frame size.

Conversely, the OPC-based communication is handled by the underlying communication driver of the real time operating system. While this protocol is widely used in industrial systems, it is aimed at facilitating rapid data sharing and access among the system's hosts. As a result, once a variable is selected for reading/writing via OPC, the update and data sharing of this variable is handled independently by the underlying communication drivers and without any interactions with the user tasks running on the controller. Therefore, user tasks that access variables via OPC do not govern the actual data exchange and they do not have knowledge on the entity that performed the changes or the time at which the variable was modified.

3.2. Control application programming languages

The IEC 61131-3 standard [41] specifies the requirements, in terms of syntax and semantics, for the programming languages used by IC. Many IC vendors have decided to make their programming systems compliant with this standard, but due to its complexity, they managed to do so up to a certain extent. The standard provides a set of programming languages, consisting of two textual languages: IL (Instruction List) and ST (Structured Text), and two graphical languages: LD (Ladder Diagram) and FBD (Function Block Diagram). A fifth language is available, called SFC (Sequential Function Chart), which acts as a framework, based on the other languages, for structuring the programs into smaller manageable units.

While it defines all aspects of the programming model, the standard also covers topics pertaining to the specific elements of the programming languages, such as: data types, variables, program organization units etc. The standard recognizes a number of elementary (pre-defined) data types (e.g. BOOL, BYTE, WORD, INT etc.), which range from 1 bit to several bytes in size, and a mechanism which allows to specify additional data types, which are in the form of structures and arrays. Variables are declared together with a data type as placeholders in the IC's memory for application-specific data. They have a certain usage (e.g. local, global, input, output etc.) within the organization unit in which they are used. The building blocks of IC applications are the program organization units (POU). Three types of POUs are defined as part of the IEC 61131-3 software model: the function (FUN), the function block (FB) and the program (PROG). These POUs can be delivered by the manufacturer, or programmed by the user. Function blocks and functions differ in the sense that they are POUs with or without memory, meaning that functions will always produce the same results given the same inputs, while function blocks need to be instantiated, they have their own static variables and their output depends on the state of their internal (memory) and external variables. Programs represent the top level of an IC's user application, acting as the "main" programs. Behaving as a regular FB, programs have the ability to access the variables assigned to physical addresses and to make them available for the other POUs.

The POUs are programmed using the programming languages mentioned before. IL is a programming language closer to machine code, resembling the assembly language. It uses one accumulator to store intermediate results. Conversely, ST is a high-level language suitable for developing complex algorithms with a syntax that is similar to the Pascal language. In FBD the IC program takes the form of function blocks, which are "wired" together in a graphical user interface to produce the information flow between the program's components. On the other hand, the LD language is derived from the relay based electrical circuit diagrams. The structure of LD considers two vertical power rails bounded by horizontal rungs implementing logic operations, which depend on the power flow reaching the connected elements. Lastly, the SFC elements provide a means of partitioning a POU into a set of interconnected steps and transitions, for the purpose of performing sequential control functions. For each step a set of actions can be defined, and each transition has a condition associated to it.

3.3. Overview of CUSUM-based ADS

Since the proposed detection algorithm embraces the cumulative sum model, we briefly review previous work and the fundamentals of this approach. We start by observing that detection algorithms which monitor the changes of variables in a time series are governed by two main parameters: the amplitude of the monitored variable, and the time elapsed between the event and its detection. The developed ADS encapsulates a detection algorithm that records the gradual change of the monitored variables, and issues an alert if the deviation is persistent. To this end, cumulative sum (CUSUM)-based approaches have been known to provide an efficient solution for this problem. They have been first proposed in [42], and presume one change-point in the mean value of the recorded time series. However, they are also capable of detecting small changes by accumulating the deviations from several samples. While the CUSUM has been used in the construction of ADS for industrial systems, throughout the literature we find several flavors that exhibit certain advantages, and possible disadvantages.

The simplest form of a CUSUM that can be computed at run-time (i.e., it can also be implemented within IC) is the one that records the changes in the parameters of an independent random series y_k. In this case it is presumed that the parameter μ_0 (the mean value) is a reference value computed in a disturbance-free scenario, while the change-point is denoted by μ_1. According to [43], the detection of the change is expressed as the change in the sign of the log-likelihood ratio. If we consider y_k, as comprising of measurements according to a Gaussian distribution, its mean value μ, and its constant variance σ^2, then the log-likelihood ratio for N observations of y_k is computed as:

$$S^N = \frac{\mu_1 - \mu_0}{\sigma^2} \sum_{k=1}^{N} \left(y_k - \mu_0 - \frac{\mu_1 - \mu_0}{2\sigma} \right), \tag{1}$$

$$m^N = \min_{1 \leq k \leq N} S^k. \tag{2}$$

According to these equations, the change is detected when $S^N - m^N \geq h$, where h is the detection threshold. Other flavors [44] of the same fundamental technique focused on the detection of positive or negative shifts using the two-sided variant and the recursive form of the cumulative sum:

$$S_{HI}^{N} = max\left(0, S_{HI}^{N-1} + y_N - \mu_0 - \frac{\mu_1 - \mu_0}{2\sigma}\right) \tag{3}$$

$$S_{LO}^{N} = max\left(0, S_{LO}^{N-1} - y_N + \mu_0 - \frac{\mu_1 - \mu_0}{2\sigma}\right) \tag{4}$$

In this case the change point is detected when $\{S_{HI}^N | S_{HI}^N \geq h\} \cup \{S_{LO}^N | S_{LO}^N \geq h\}$.

Similarly, Nadler and Robbins [45] proposed a variant able to detect a change in μ in either directions if $R^N \geq h$, where:

$$R^N = \max_{1 \leq j \leq N} \sum_{k=1}^{j} (y_k - \mu_0) - \min_{1 \leq j \leq N} \sum_{k=1}^{j} (y_k - \mu_0) \tag{5}$$

We observe that if multiple change points need to be detected, the above-mentioned methods must set the next starting point of the cumulative sum to the previous changing point. However, if multiple successive changes are present, the cumulative sum is frequently restarted, which increases the detection time. Furthermore, if the change of monitored variables is persistent (e.g., the attack runs over a longer time period), then this also changes the mean computed value, which significantly delays the detection of the absence of the disturbance. We note that an intrinsic novelty of the proposed technique is that it facilitates the restoration of the normal (i.e., attack-free) operating state. For this purpose, it builds on the cumulative sum technique to detect both the start and the end of disturbances.

3.4. Proposed ADS architecture

The design of the proposed ADS must account for several restrictions. While apparently the hardware of modern IC may provide sufficient computing power to run complex detection algorithms (e.g., neural networks, clustering techniques) as a distinct OS process (i.e., on top of the OS), this strategy brings a significant competitor to the real-time scheduler for the available processing time. Subsequently, it can cause significant delays in the scheduling of critical tasks, and, ultimately, it can halt the scheduler (a behavior that is triggered automatically when a task cannot be scheduled within a specific time period - configured with the help of a "watchdog timer"). Besides these aspects, normally, control application developers do not have access to the underlying OS. This restriction stems from the immense risk of altering the correct behavior of the real-time scheduler when uncontrolled changes are performed to the OS. Consequently, developers must use the available programming models and languages (detailed in the previous sections), which significantly reduce the size and complexity of the applicable instruction set.

According to these observations, it is clear that any practical proposal for an ADS for IC needs to be positioned in the user tasks and it must account for the resources available at this layer. Therefore, in order to ensure the practical integration of the developed ADS into existing control applications, we further identify two fundamental requirements: (i) the ADS implementation must be modular and it needs to be isolated from the existing control logic in order to minimize the required changes and to reduce the complexity of maintaining its code (e.g., debugging, updating); and (ii) the anomaly detection algorithm must account for the limited programming features available at this level (e.g., adopt simple arithmetic operations), while ensuring that the schedulability of user tasks is not affected.

The architecture of the proposed ADS addresses the two aforementioned requirements as follows. Firstly, the detection algorithm is implemented as a separate user task, hereinafter called the *monitoring* task. This is a periodic task configured with the lowest priority, which repeatedly records the task's start time and runs a lightweight anomaly detection algorithm. The period of the monitoring task is chosen as low as possible in order to ensure high sensitivity to scheduling delays caused by an increasing number of interrupts (e.g., input events, increased number of packets received on a networking interface), or by the abnormal behavior (e.g., change of load) of other user tasks. An example execution of the proposed monitoring task has been included as part of Figure 2. Here, we observe that the monitoring task is configured with the lowest priority (among user tasks) in order to ensure that its scheduling time is influenced by all the planned tasks (through subsequent interruptions and delays). As a result, a careful recording of the delays occurring in the scheduling time of the monitoring task can indicate the change of behavior in the application's execution. This can further indicate the presence of internal/external disturbances.

Secondly, the monitoring task implements a detection algorithm that leverages the sensitivity of the scheduling time of user tasks to disturbances in order to detect abnormal application behavior. The detection algorithm is formulated in terms of simple arithmetic operations in order to minimize the execution overhead and to guarantee that schedulability is not affected. In order to formalize the schedulability condition we introduce the following notations. Let $\Gamma = \{\gamma_1, ...\gamma_n\}$ be the set of regular (i.e., control) user tasks and $\gamma_m \notin \Gamma$ the monitoring task. Then, let w_i denote the worst case execution time, T_i the configured period, and d_i the deadline of the i-th task. Let w_m denote the worst case execution time, T_m the period, and d_m the deadline of the monitoring task. In order to guarantee schedulability, the following conditions are formulated.

First, the average system utilization rate must not exceed a certain upper bound μ. By presuming the well-known rate monotonic scheduling algorithm [46], the schedulability condition is formulated as:

$$\frac{w_m}{T_m} + \sum_{i=1}^{n} \frac{w_i}{T_i} < \mu \tag{6}$$

Since the monitoring task has the lowest priority, it can be preempted by any other task. However, if the monitoring task is not scheduled by a particular deadline, the system (i.e., scheduler) halts with an error that signals the missing deadline. To address this issue, the following condition needs also to be satisfied:

$$w_m + \sum_{i=1}^{n} \left\lceil \frac{w_i}{T_i} \right\rceil w_i < d_m \tag{7}$$

3.5. Proposed detection algorithm

As already stated, the detection algorithm is implemented as a separate (monitoring) task. Its operation builds exclusively on the information available to this task. It leverages the measured start time of the task, which denotes the time at which the task begins its execution. Let t_j be the j-th start time of the monitoring task, and $\widetilde{T}_j = t_j - t_{j-1}$ the j-th measured task period, that is, the time elapsed from the previous run. We note that for a high priority task the measured period has a mean value close to the configured task period, while exhibiting a slight variation (i.e., jitter) due to the system interrupts. Conversely, a task with the lowest priority has a significantly higher jitter for the measured task period.

In order to minimize the computational overhead and to ensure the proposal's practical integration in existing (legacy) controllers, the starting point in the construction of the proposed detection algorithm is the statistical cumulative sum. The cumulative sum (CS_j) is computed as the difference between the measured task period (denoted by $\widetilde{T}_j$) and the expected task period value (denoted by $\hat{T}$):

$$CS_j = CS_{j-1} + (\widetilde{T}_j - \hat{T}), \quad \text{where} \quad CS_0 = 0 \tag{8}$$

Next, by using a moving window technique we approximate the cumulative sum with a regression line for a time window consisting of N samples. This is particularly useful in the current scenario since the slope's modification (i.e., upward or downward trend in the cumulative sum) can indicate an intervention on the system. The time window is divided into N equal points $x_k, k = \overline{1, N}$. Then, the cumulative sum is approximated by a linear segment:

$$CS = \beta * x + \alpha, \tag{9}$$

where:

$$\beta = \frac{\sum_{k=1}^{N} \left[(x_k - \overline{x})(CS_k - \overline{CS})\right]}{\sum_{k=1}^{N} (x_k - \overline{x})^2}, \alpha = \overline{CS} - \beta * \overline{x} \tag{10}$$

In Eq. (10) $\overline{x}$ is the average of x, and $\overline{CS}$ is the average of the cumulative sum's values in the selected time window. Based on these equations, the anomaly detection system uses the value of β for detecting an abnormal behavior. To this end, if β is found outside a given interval $[-\delta, \delta]$, an alert is generated (where δ is the detection threshold). The computation of the detection threshold δ is performed according to the recommendations set forth in the field of statistical analysis [47]. Additional details are provided later in Section 4.2.

4. Implementation

4.1. Infrastructure

The implementation has been mainly evaluated in the context of a testbed that recreates the main components of a typical gas distribution node from a Romanian natural gas transportation network (see Figure 3). The system builds on a primary controller (PLC^P) produced by Phoenix Contact, model ILC 350-PN. PLC^P runs the necessary control logic and handles the communication (OPC, Modbus TCP, Modbus RTU) with the other components, including the secondary controllers (PLC^S), the Modbus RTU slaves, HMIs, which are all typically found in the automation solution of the considered gas distribution node.

Given the significance of the primary controller, most of the analysis and results documented in this paper are focused on this controller. With the help of a local automation company specializing in natural gas transportation systems, PLC^P has been configured with several tasks and programs in order to replicate its typical load from a production environment. Next, the configured tasks are described according to the tuple (period, deadline, [best case - worst case execution time], priority). To this end, PLC^P runs the following periodic tasks: a control task (100ms, 200ms, [10ms 30ms], 1); one task for reading inputs (50ms, 100ms, [1ms 10ms], 1); a Modbus/RTU task (10ms, 100ms, [1ms 2ms], 0); a Modbus/TCP task (100ms, 500ms, [1ms 2ms], 2), and the proposed monitoring task (10ms, 200ms, [1ms 1ms], 3).

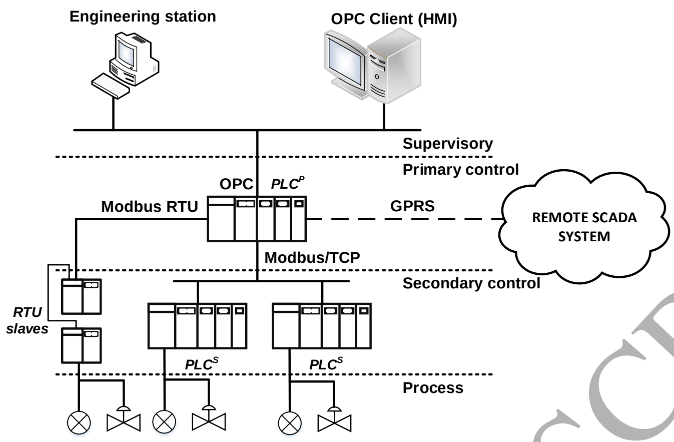


Figure 3: Laboratory testbed architecture that recreates a gas distribution node from a Romanian gas transportation network.

4.2. Detection implementation details

The implementation was carefully tailored to address the limited computational resources of application-layer programs. The main objective was to reduce the required memory, and to define the proposed detection algorithm in terms of simple arithmetic operations supported by the IC's application-layer programming languages.

We start by observing that the main consumption of memory resources stems from the need to store N samples of (x_k, CS_k), used to compute the value of β in Eq. 10. However, the value of N can grow significantly if we presume that the monitoring task is scheduled to run with a low period (e.g., every 5 ms). Therefore, to reduce the number of stored samples, the implementation stores only every q-th value of the cumulative sum. It should be noted that, this way, measurements are not lost, since they are still aggregated into the value of CS.

We observe that the main computational cost of Eq. 10 is the requirement to continuously store and compute the average CS values. In order to eliminate its computation, and to simplify the implementation of Eq. 10, the regression strategy is applied on a symmetric time interval $[-p, p]$. Accordingly, we have that $N = 2p + 1$. As a result of the above-mentioned changes, Eq. 10 can be rewritten as:

$$\beta = \frac{3 * \sum_{k=1}^{2p+1} (CS_{q*k})(k-p-1)}{p(p+1)(2p+1)}, \tag{11}$$

It should be noted that Eq. (11) yields a much simpler implementation compared to Eq. (10). Furthermore, by observing that p is known apriori, then one may precompute $\frac{3}{p(p+1)(2p+1)}$ and store its value as the constant C_1, which further simplifies the computation of β:

$$\beta = C_1 * \sum_{k=1}^{2p+1} (CS_{q*k})(k-p-1). \tag{12}$$

Besides these aspects, if we presume that a circular buffer $\mathcal{B}$ is used to store the elements of CS_{q*k} ($k = \overline{1, 2p+1}$), then the implementation of Eq. (12) reduces to the repeated multiplication of two elements from two distinct buffers. A first buffer $\mathcal{B}$ is the circular buffer that holds the values of CS_{q*k}. A second buffer is used to store the individual values of $C_2^k = (k-p-1)$. Consequently, the computation of β is further reduced to the following equation:

$$\beta = C_1 * \sum_{k=1}^{2p+1} C_2^k \mathcal{B}_{(ptr+k)\ mod\ (2p+1)}, \tag{13}$$

where ptr is a modulo $2p+1$ pointer that identifies the current write position in the circular buffer.

According to these simplified equations, the implementation of the proposed detection algorithm is summarized as Algorithm 1, while the computation of β is included in Algorithm 2. As listed in Algorithm 1, at each call, the system tick is recorded as t_j, which is followed by the computation of $\widetilde{T}_j$ and of CS_j according to the previous equations. Then, if q measurements have been recorded, the aggregated value of CS_j is stored in the circular buffer $\mathcal{B}$. The value of the slope (β) is then determined according to Algorithm 2. Once the value of β is computed, an anomaly is signaled in case β does not fall within the bounds of

Algorithm 1: Implemented Anomaly Detection Algorithm

```
   Input  : SYSTEM_TICK, $\hat{T}, q, j, \delta, \mathcal{B}, ptr$
   Output: ANOMALY_ALERT
1  $t_j$ := SYSTEM_TICK;
2  $\widetilde{T}_j := t_j - t_{j-1}$;
3  $CS_j := CS_{j-1} + (\widetilde{T}_j - \hat{T})$;
4  if $j \ mod \ q == 0$ then
5      $\mathcal{B}_{ptr} = CS_j$;
6      $\beta$ := @ComputeSlope($\mathcal{B}, ptr$);
7      if $\beta \notin [-\delta, \delta]$ then
8          ANOMALY_ALERT:=1;
9      else
10         ANOMALY_ALERT:=0;
11     end
12 end
13 $j := j + 1$;
14 $ptr := (ptr + 1) \ mod \ (2p + 1)$;
```

Algorithm 2: Implemented β Computation Algorithm

```
   Input  : $\mathcal{B}, ptr$
   Output: $\beta$
1  sum := 0;
2  for $k := \overline{1, 2p+1}$ do
3      sum := sum + $C_2^k * \mathcal{B}_{(ptr+k) \ mod \ (2p+1)}$;
4  end
5  $\beta$ = sum $* C_1$;
```

the interval $[-\delta, \delta]$.

The algorithm can be further refined with several levels of criticality, which would permit different set of actions to be performed for each distinct level. Accordingly, we pursue this direction by defining two detection intervals via two threshold values $\delta_{\texttt{ALERT}}$ and $\delta_{\texttt{ERROR}}$, where $\delta_{\texttt{ALERT}} < \delta_{\texttt{ERROR}}$. The ALERT level corresponds to a minor deviation from the normal behavior, while the ERROR level denotes a major deviation from the normal behavior, which would require immediate intervention. We define the two levels according to the recommendations set forth in the field of statistical analysis [47]. By using β's standard deviation σ_β, we define $\delta_{\texttt{ALERT}} = 3\sigma_\beta$, which means that 99.7% of measurements fall within the "normal" distribution, and a more restrictive threshold of $\delta_{\texttt{ERROR}} = 5\sigma$, which denotes abnormal behavior that is outside the typical distribution.

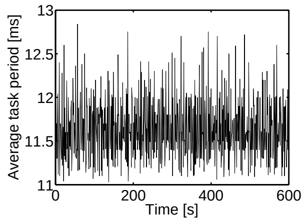
5. Experimental results with the ILC 350-PN controller

This section documents the results produced using the ILC 350-PN controller in the context of the natural gas transportation system testbed described in the previous section. The analysis focuses extensively on the network traffic, since, as demonstrated later, traditional computer scans have a significant impact on the industrial traffic, and ultimately, on the scheduling rate of program tasks. We start by illustrating the recorded task rate values and the network traffic in the case of the intervention-free scenario. Then, we proceed with the results in the context of operator-specific interventions: downloading new program sources to the IC via the OPC protocol, and uploading FTP files to the IC's flash card. Lastly, we illustrate the impact of network discovery tools (i.e., `nmap`) on the industrial traffic and we thoroughly analyze the sensitivity and the effectiveness of the developed technique with various configurations.

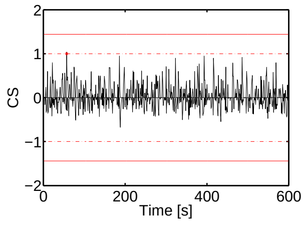
The analysis showcases not only the performance of Algorithm 1, but it also compares the solution with the traditional CUSUM algorithm. In the later case we presume two threshold levels, and, for the clarity of presentation, we use a different symbol h to identify the levels for the traditional CUSUM algorithm, whereas δ denotes the threshold levels for β (the slope of the linear segment in the proposed detection algorithm). To this end, we leverage the following notation to distinguish the two threshold levels for the traditional CUSUM algorithm: $h_{\texttt{ALERT}} = 3\sigma_{CS}$, and $h_{\texttt{ERROR}} = 5\sigma_{CS}$, where σ_{CS} is the standard deviation of the CUSUM computed according to Eq. (8). The implementation on the ILC 350-PN has been performed according to the IEC 61131-3 specifications by using the Structured Text language.

5.1. Intervention-free operation

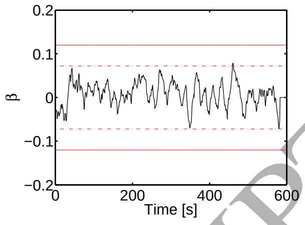
We begin with the analysis of the proposal's parameters and of the network traffic in the context of the intervention-free scenario. The results have been



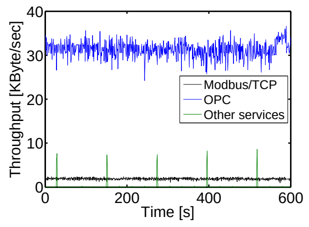
(a) Average task period.



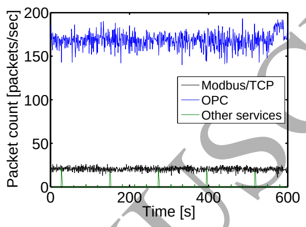
(b) CS_j.



(c) β.



(d) Traffic throughput.



(e) Packet count.

Figure 4: Recorded measurements in the intervention-free scenario.

summarized in Figure 4. The dashed red lines in Figures 4b and 4c denote the `ALERT` thresholds for $h_{\texttt{ALERT}}$, and $\delta_{\texttt{ALERT}}$, respectively, while the continuous red lines in the same two figures denote the `ERROR` thresholds for $h_{\texttt{ERROR}}$, and $\delta_{\texttt{ERROR}}$, respectively.

Here, we observe that the monitoring task's average period (computed over 1s) exhibits a slight variation between 11ms and 12.76ms (see Figure 4a). Nevertheless, we further observe that its period also exhibits a normal distribution, as depicted in Figure 5. Accordingly, we measured a mean task period value of 11.63ms, a standard deviation of 0.346ms, and a maximum value of 12.76ms. A significant aspect to observe is that, even in the intervention-free scenario, the expected value of the monitoring task period is larger than 10ms (the configured task period). This is owed to the lowest task priority.

In terms of detection, in this scenario, both the CUSUM-based and the β-based computations reported an `ALERT`. In case of CUSUM, an `ALERT` is reported at 57s, while in case of the β computation an `ALERT` is reported at 460s. These fall within the boundaries of the permitted 0.3% of measurements (according

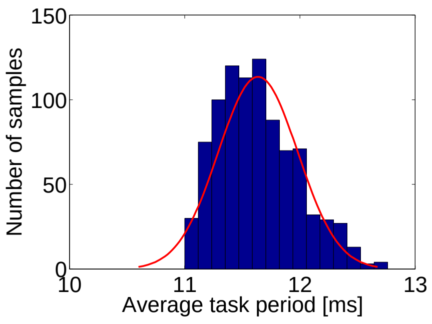


Figure 5: The distribution of average task periods on PLC^P.

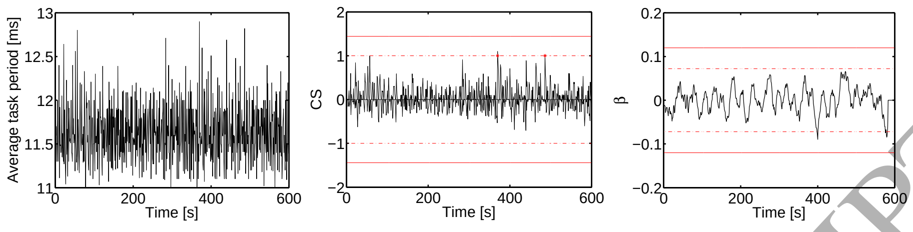
to the 3σ threshold value). However, we further note that these "outliers" are singular and fall within the "normal" statistical distribution of the monitoring task's period.

Lastly, the recorded traffic shows an average throughput of 32KBps for the OPC traffic between the OPC server and PLC^P, and an average of 1.88KBps of Modbus/TCP traffic between PLC^P and PLC^S, and few "outliers" for other traffic (specific to the ILC PN-350). As later demonstrated, a significant aspect is represented by the number of packets, where we observe an average of 168 packets/sec for the OPC traffic, and an average of 20 packets/sec for Modbus/TCP. These are significant details which are used later as a metric for measuring the impact of traditional network scanning attacks.

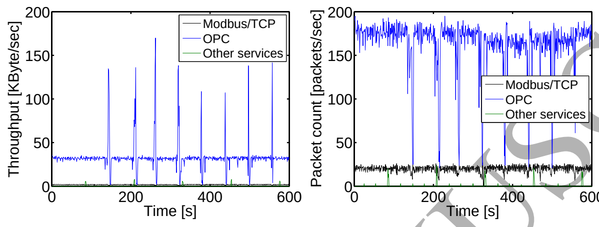
5.1.1. Regular interventions

Next, we analyzed the performance of the developed detection algorithm in the context of typical interventions that are regularly performed by system engineers. To this end, we measured the behavior of the measurement task in the context of the following interventions: (i) downloading program sources to PLC^P via Phoenix Contact's PC Worx programming environment; and (ii) downloading a file to PLC^P's flash memory via FTP.

For the first intervention scenario the measured results are summarized in Figure 6. Here, we observe the changes in the network throughput, especially in the case of the OPC protocol, each time a download is triggered. An interesting



(a) Average task period. (b) CS_j. (c) β.



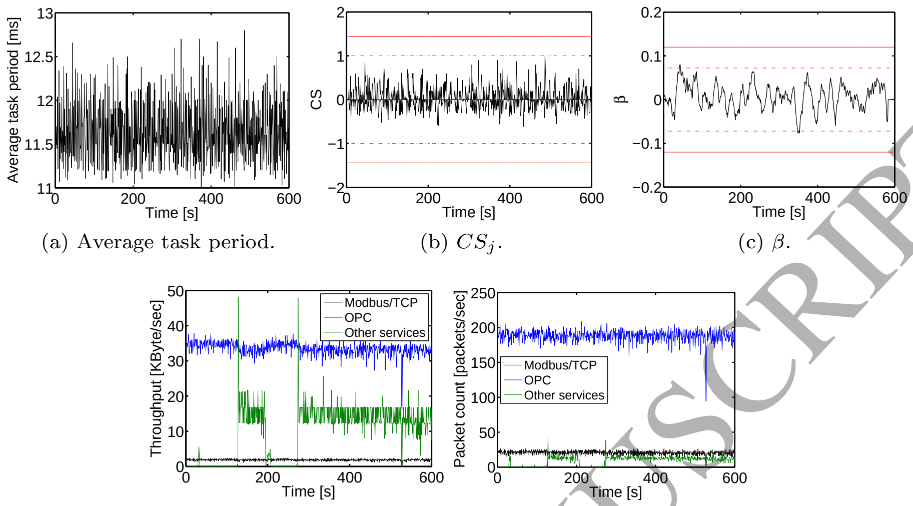
(d) Traffic throughput. (e) Packet count.

Figure 6: Recorded measurements in the case of new sources downloaded to the PLC via Phoenix Contact's PC Worx programming environment.

aspect that was observed throughout our measurements with respect to the functioning of the ILC 350-PN is that the controller does not increase the packet count while the download is in progress, but it increases the packet sizes instead, and, as a result, the traffic throughput.

In terms of detection, similarly to the intervention-free scenario, the `ALERT` level is reached only in few singular cases, which fall within the "normal" statistical distribution of the monitoring task's period. The explanation for this behavior, and for the fact that the task period is not affected by this intervention, derives from the analysis of the network traffic. As already mentioned, the download process decreases the number of packets, which, essentially, reduces the number of OS interrupts for processing packets. As a result, the download procedure does not convey an additional load to the controller, as one may expect, but it is actually governed by the OS in order to minimize the impact on the IC's internal operation.

The measured results for the second intervention are summarized in Figure 7. In this case two distinct files have been downloaded, highlighted by a green



(a) Average task period. (b) CS_j. (c) β.

(d) Traffic throughput. (e) Packet count.

Figure 7: Recorded measurements in the case of two FTP downloads to the PLC's flash card.

line in Figure 7d. The results further confirmed the previous measurements, where the ILC 350-PN regulates the traffic so as the task scheduling would not be affected by the regular packet exchanges. Furthermore, we observed that when several parallel FTP sessions are initiated, the controller reduces the overall FTP throughput in order to limit the impact on the scheduled tasks.

5.1.2. Network discovery attempts

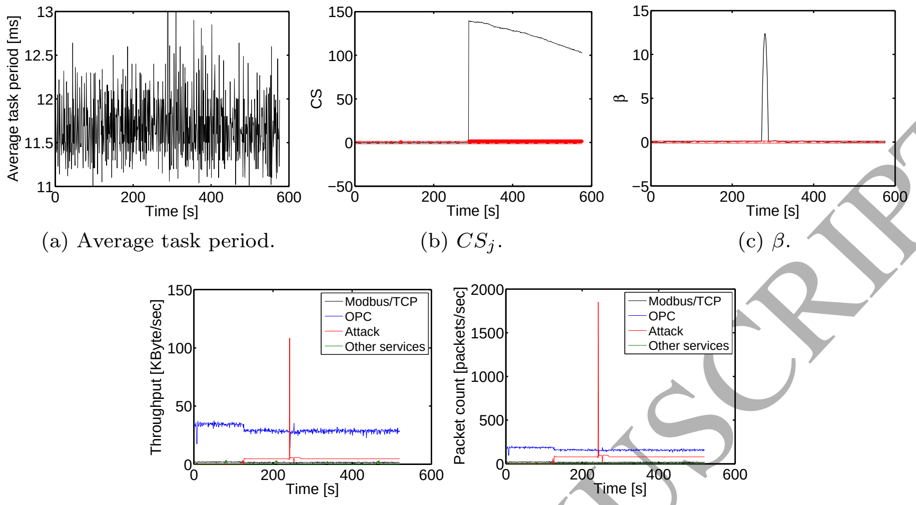
While the regular exploitation of the IC's services (e.g., FTP) did not affect the network traffic and the scheduling of tasks, the use of traditional network scanning tools may significantly affect the controller's internal operation. Accordingly, in the next step, we launched a series of scanning attacks against PLC^P. For this purpose we launched the `nmap` tool with its default configuration to perform SYN and UDP scanning against the IC. The results have been summarized in Figure 8.

If we look at the distribution of the measurement task periods in Figure 8a, the changes are barely visible. In this case, the mean task period increases to 11.92ms (from 11.63ms in the intervention-free scenario), which yields a deviation of 2.43%. However, by leveraging the CUSUM-based computations, the detection algorithm reports an `ALERT`, followed by a repeated number of `ERROR` events (see Figure 8b). This is owed to the `nmap` tool, which, at a particular moment (in 1s), issues a large number of packets (see Figures 8d and 8e) that increase the measured task period. We observe that in this case the computed CS_j does not return to its "normal" value not even after 5 minutes.

Conversely, in case we adopt the proposed detection algorithm with distinct change points, the value of T_{avg} is periodically recomputed. Accordingly, the algorithm can pin-point the start and end moments of the attack (see Figure 8c) and triggers `ALARM` and `ERROR` alerts accordingly. This is a salient feature of the proposed detection algorithm that distinguishes itself from the existing CUSUM-based approaches by being capable to precisely indicate the start and end of external disturbances. This brings significant advantages, since it can help the trouble-shooting procedures by indicating the attack interval, which can be correlated with the output of other network/host-based detection systems in order to diagnose each particular event.

A notable aspect of these measurements is that they showed an undocumented behavior of the ILC 350-PN controller, which is otherwise widely used in the Romanian gas transportation network. More specifically, the measurements showed the sensitivity of the controller to non-governed traffic, that is, to a flow of network packets that is not regulated by its internal services and packet processing algorithms. While the experiments from the previous sections have failed to disturb the controller's internal operation via its regular services (even with parallel FTP downloads), we observe that by simply increasing the number of packets, the internal operation of the ILC 350-PN is significantly affected.

Obviously, the network scan in this case was rather intrusive, since `nmap` issued almost 2000 packets/sec at 242s. However, to further understand the sensitivity of the controller to the number of received packets, we performed additional experiments. Accordingly, we used the `tcpreplay` tool to replay a previously recorded `nmap` traffic at specific packet rates. Two distinct rates have



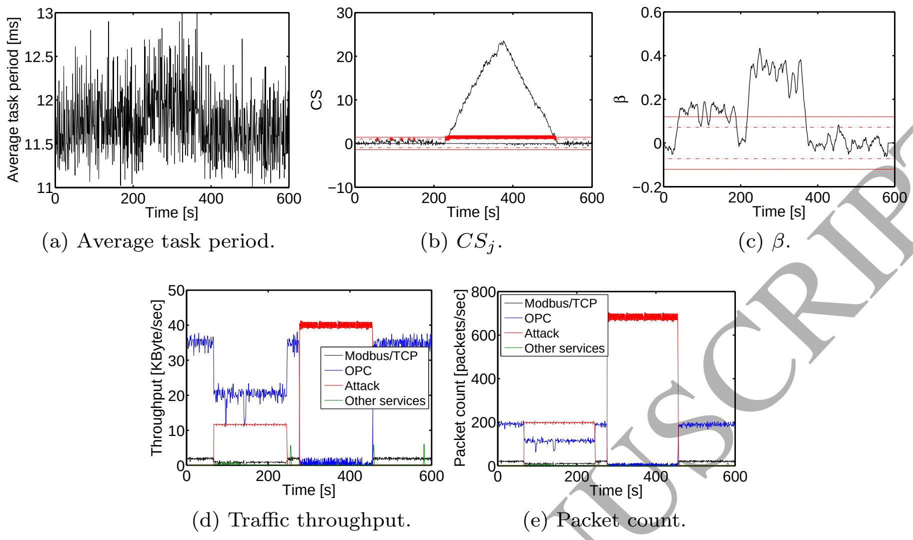
(a) Average task period. (b) CS_j. (c) β.

(d) Traffic throughput. (e) Packet count.

Figure 8: Recorded measurements in the case of an `nmap` scan attack against PLC^P.

been considered: 200 packets/sec, which is comparable to the OPC packet count, and a more intrusive scenario with 700 packets/sec. As demonstrated by the results in Figure 9, the industrial traffic (both the OPC and Modbus/TCP) has been significantly affected by the network scan. To this end, the industrial traffic is reduced from an average of 200 packets/sec to an average of 120 packets/sec in the case of the 200 packets/sec attack. Conversely, the 700 packets/sec network scan, while it has a similar throughput with the industrial traffic (40KBps), it is able to reduce the OPC and Modbus/TCP traffic to 0 KBps. This essentially interrupts the communication between the OPC server and PLC^P, as well as the communication between PLC^P and PLC^S.

However, we observe that in this particular case both the CUSUM-based as well as the proposed algorithms are able to successfully detect the two distinct attacks. Nevertheless, as demonstrated by the results, the CUSUM does not distinguish between the two attacks. Conversely, by leveraging the proposed algorithm, the computation of β precisely distinguishes the two distinct attacks. While these results provide further evidence on the effectiveness of the developed



(a) Average task period. (b) CS_j. (c) β. (d) Traffic throughput. (e) Packet count.

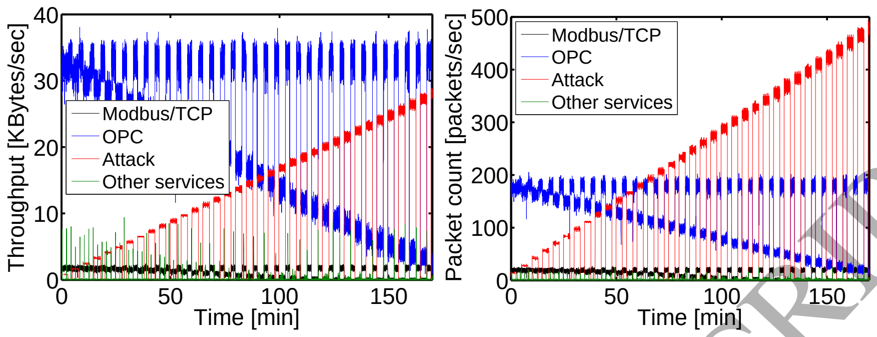
Figure 9: Recorded measurements in the case of two replay attacks against PLC^P: 200 packets/sec and 700 packets/sec.

detection algorithm, they raise serious alarms in terms of leveraging specific controllers for critical control systems. As demonstrated in this section, the ILC 350-PN controller, while it is considered a high-end system, it is highly susceptible to network scans, and its internal operation can be significantly altered via traditional network discovery attacks.

5.2. Sensitivity analysis

We further analyzed the sensitivity of the proposed algorithm in terms of true positive rates (TPR) and false positive rates (FPR). The assessment focused on the parameters that affect the sensitivity of the proposed algorithm: task period and attack rate. For the purposes of this experiment we launched a gradual attack of 178 minutes with the help of the `tcpreplay` tool. The attack gradually increased the packet rate from 14 packets/sec up to 500 packets/sec.

As illustrated in Figure 10, the controller is highly sensitive to non-regulated packets, and the impact on the industrial traffic is visible even at low packet rates (e.g., 50 packets/sec). To this end, the industrial traffic throughput is re-



(a) Traffic throughput. (b) Packet count.

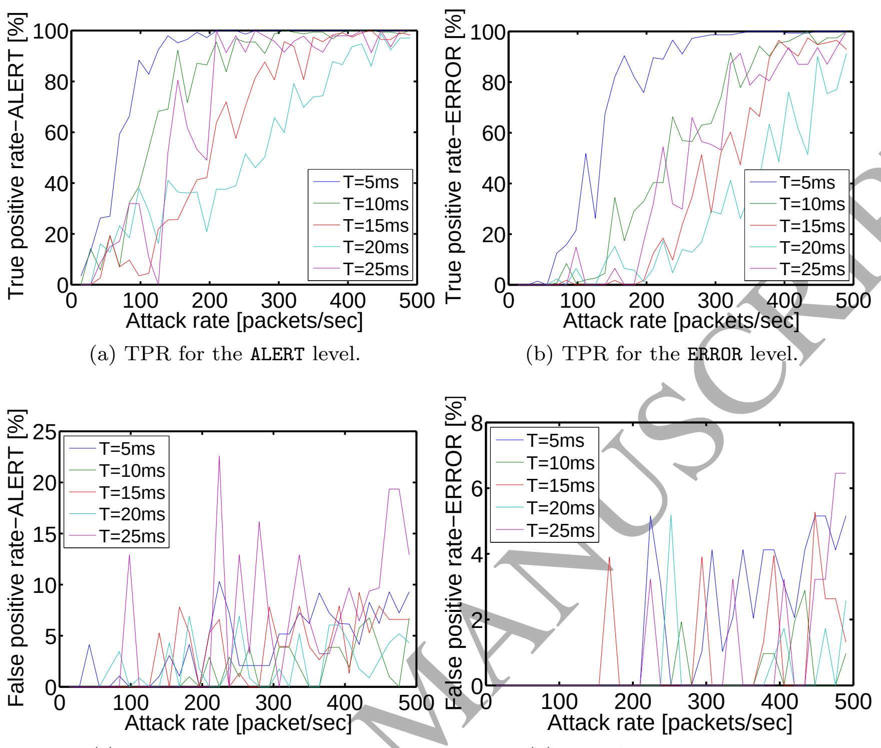
Figure 10: Gradual packet replay attack against PLC^P.

Table 1: True positive rates (TPR) for `ALERT` and `ERROR` levels.

Rate	ALERT					ERROR				
[p/s]	5ms	10ms	15ms	20ms	25ms	5ms	10ms	15ms	20ms	25ms
14	3.42	0	0	0	0	0	0	0	0	0
28	13.01	14.10	0	0	0	0	0	0	0	0
56	26.89	19.35	19.29	12.71	14.89	0	0	0	0	0
84	66.20	32.25	9.64	18.39	31.91	15.86	8.38	1.75	0	0
140	97.91	69.03	25.43	41.04	52.17	67	4.51	0	8.67	0
210	100	95.45	63.71	37.57	100	89.58	40.25	12.38	6.35	31.91
308	100	100	95.57	59.88	91.48	98.61	73.85	52.21	27.9	53.19
500	100	100	98.21	97.05	100	100	100	92.85	91.17	100

duced significantly with an attack of 100 packets/sec, while the 500 packets/sec attack fully interrupts regular communications. A notable aspect of these measurements is that, compared to the regular traffic, the attack traffic throughput is significantly lower. Accordingly, as depicted in Figure 10a, a 20KBps attack reduces the regular traffic from 33KBps to 10KBps, while a 30KBps attack causes the interruption of all communications with PLC^P. This is an important aspect that confirms, once again, the sensitivity of industrial controllers to external interventions. Furthermore, this information can represent a significant asset in the hands of malicious actors trying to mount cyber stealth attacks against industrial systems [48].

In terms of the values for TPR and FPR we conducted an extensive set



(a) TPR for the `ALERT` level.

(b) TPR for the `ERROR` level.

(c) FPR for the `ALERT` level.

(d) FPR for the `ERROR` level.

Figure 11: True positive rates (TPR) and False positive rates (FPR).

of measurements with various periods configured for the monitoring task, and different attack rates. The analysis focused on the two detection levels: `ALERT` and `ERROR`. Accordingly, we selected five distinct task periods (5ms, 10ms, 15ms, 20ms, and 25ms) for the monitoring task, and in each case we launched the gradual attack against PLC^P. The results have been summarized in Figure 11 and numerically in Table 1.

The TPR for the `ALERT` level shown in Figure 11a proves that a lower task period (i.e., of 5ms) increases the sensitivity of the detection engine. In this case the TPR increases from 3.42% for 14 packets/sec to 66.2% for 84 packets/sec and up to 100% for 210 packets/sec. By increasing the task period to 10ms, the sensitivity of the detection algorithm slightly decreases. To this end, for

Table 2: False positive rates (FPR) for `ALERT` and `ERROR` levels.

Rate	`ALERT`					`ERROR`				
[p/s]	5ms	10ms	15ms	20ms	25ms	5ms	10ms	15ms	20ms	25ms
14	0	0	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0	0
56	0	0	0	0	0	0	0	0	0	0
84	1.03	0	0	3.44	0	0	0	0	0	0
140	1.02	0	5.26	0	0	0	0	0	0	0
210	5.15	2.88	5.26	0	0	0	0	0	0	0
308	5.15	3.84	3.94	1.72	4.12	0	0	0	0	0
500	9.27	6.73	6.57	4.31	12.9	5.15	0.96	1.31	2.58	6.45

14 packets/sec the algorithm is unable to detect the attack (i.e., TPR=0%). In case the attacker increases the packet rate to 28 packets/sec the TPR also increases to =14.1%. Obviously, by further increasing the packet rate, we also measure an increased TPR, which reaches 100% at 308 packets/sec.

Next, we look at the TPR for the `ERROR` level. In this case the results are consistent with the previous analysis. Accordingly, a lower period configured for the monitoring task yields a higher precision, while selecting a higher task period decreases the precision of the detection algorithm. Besides these aspects we note that the value of the TPR is rather low in the 14-56 packets/sec attack interval because the value of β only reaches the `ERROR` threshold above these packet rates. Starting with 84 packets/sec the algorithm triggers `ERROR` events and the value of the TPR reaches 15.86% in the case of the 5ms task period. The TPR further increases to 100% for 500 packets/sec.

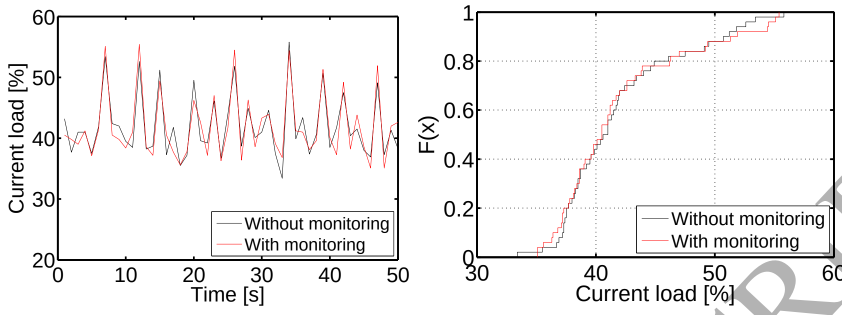
In case of the FPR (see Figure 11 and Table 2), for both levels we recorded reduced values. Accordingly, as shown in Figures 11c and 11d the FPR exhibits small variations between 1 and 10% for the `ALERT` level and for packet rates below 100. The variations slightly increase above the 100 packets/sec rate up to 20% in singular cases. However, in the majority of cases we measured an average FPR below 6%. For the `ERROR` level, the value of the FPR is even lower, and its variations are measured only above 150 packets/sec. Furthermore, in this case the maximum recorded FPR did not exceed 6.45%.

It should be noted that, while the analysis performed in this section is focused on the packet rate analysis, for illustration purposes, the actual measurements and attack analysis have been performed by leveraging external hardware and direct packet capturing (i.e., port mirroring and the `tcpdump` tool). However, recall that, as previously stated, IC applications do not have direct access to network traffic, and packet exchanges are not visible at this level. Therefore, the use of traditional metrics such as packet count and traffic throughput, as part of an application-level detection methodology, would not represent a practical solution. Conversely, by carefully monitoring the scheduling rate of user tasks, the developed methodology can detect any changes that may ultimately affect the scheduler and the execution of user tasks.

Lastly, we conclude that, while the proposed detection algorithm exhibits a good performance in the case of a reduced task period, the use of a low task period also increases the load on the industrial controller. Accordingly, in control programs only critical tasks are scheduled to run at a 5ms period, and a large number of critical tasks can significantly affect the controller's performances and can endanger schedulability. Therefore, based on the above measurements, we believe a good compromise in terms of the practical integration of the proposed detection engine in production environments and performance, is to configure a slightly higher task period. In the analyzed system, a good trade-off between the two aspects is obtained by the 10ms task period. However, for other systems an analysis is needed in order to ensure that the controller's performance is not affected by the integration of the proposed detection engine.

5.3. Impact on schedulability

A significant dimension of the proposed detection methodology is its impact on schedulability. According to Eq. (6), the system load (including all tasks and programs) should not exceed the value of μ in order to guarantee schedulability. Based on the work reported in [46], in general, the value of μ should not exceed 0.7 to guarantee schedulability. In order to assess the impact on schedulability, we recorded the average system load for each second of operation for 10 hours.



(a) Measured system load. (b) Computed CDF.

Figure 12: Impact of the proposed detection methodology on schedulability.

For illustration purposes Figure 12a shows the system load for a 50s time window. Here, it is shown that, without the monitoring task, the system load (as simulated via the three periodic tasks) does not exceed 0.6. Furthermore, by activating the monitoring task the system time exhibits an insignificant variation. This means that these deviations are part of the same distribution and that the monitoring task has an insignificant impact on the system load. To further confirm this aspect we use the Kolmogorov-Smirnov statistical test (K-S test) [49]. The null hypothesis is that the recorded system load in the absence and in the presence of the monitoring task is part of the same statistical distribution. The cumulative distribution function computed on the two distinct measurements is shown in Figure 12b. The result of the test is 1 if the test rejects the null hypothesis at the 5% significance level, and is 0 otherwise. By applying the K-S test on the measured system load, the acceptance of the null hypothesis was confirmed.

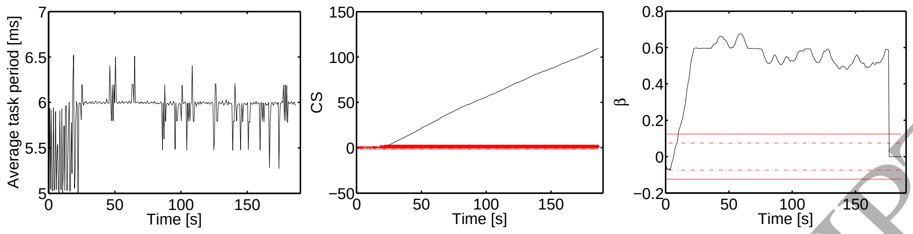
6. Experimental results with the Siemens SIMATIC S7-1200 Programmable controller

In order to demonstrate the feasible integration of the proposed detection engine into other IC, we performed a series of measurements with the Siemens SIMATIC S7-1200 Programmable controller (S7-1214C CPU). The testbed consisted of the S7-1200 IC, an engineering station running the Totally Integrated Automation portal (the TIA portal), an OPC server, and an Ethernet switch.

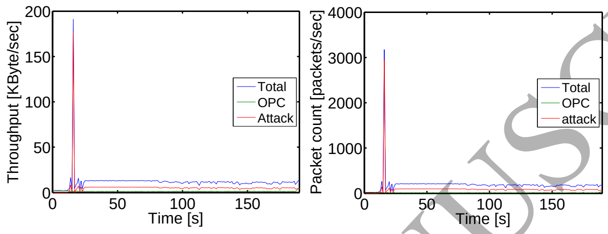
In terms of its software architecture, the S7-1200 family of controllers are notably different from the previously described model. The controller has one single scan cycle for writing to the output ports, for reading the input ports, for sequentially executing the organization blocks (OB) from the user program, and for processing interrupts at any moment during the scan cycle. An important aspect, which also distinguishes it from the ILC 350-PN controller's operation model, is that communication requests are handled periodically throughout the scan cycle by interrupting the user program. The controller supports up to four cyclic interrupt events with a higher priority than that of the default scan cycle. The cyclic interrupts have user defined organization blocks attached for running the user programs. The system guarantees that the scan cycle will be completed in a predefined maximum cycle time; otherwise, a time error event is generated.

According to the S7-1200 controller's architecture, to simulate a variable load similar to a controller running in a production environment, we defined three tasks (as in the experiments conducted in the previous section) organized in 3 cyclic interrupt organization blocks. The proposed detection algorithm was implemented as a program block executed within the main scan cycle, thus with the lowest possible priority. This configuration permitted the integration of the proposed detection algorithm according to our initial requirements, where the monitoring code needs to run with the lowest priority in order to record the scheduling deviation of all the other tasks.

The implementation of the detection algorithm on the S7-1200 controller was made in Ladder Logic. A special attention was given to the configuration of the communication load (i.e., the percentage of the time dedicated to the communication tasks). Compared to the ILC 350-PN, the S7-1200 controller can set the maximum load attributed to communications as a percentage of the maximum cycle time. Accordingly, in our implementation the maximum cycle time was set to 200ms, the communication load was set to 20%, the cyclic interrupt event queue was set to 1, and the enable time error option was activated.



(a) Average task period. (b) CS_j. (c) β.

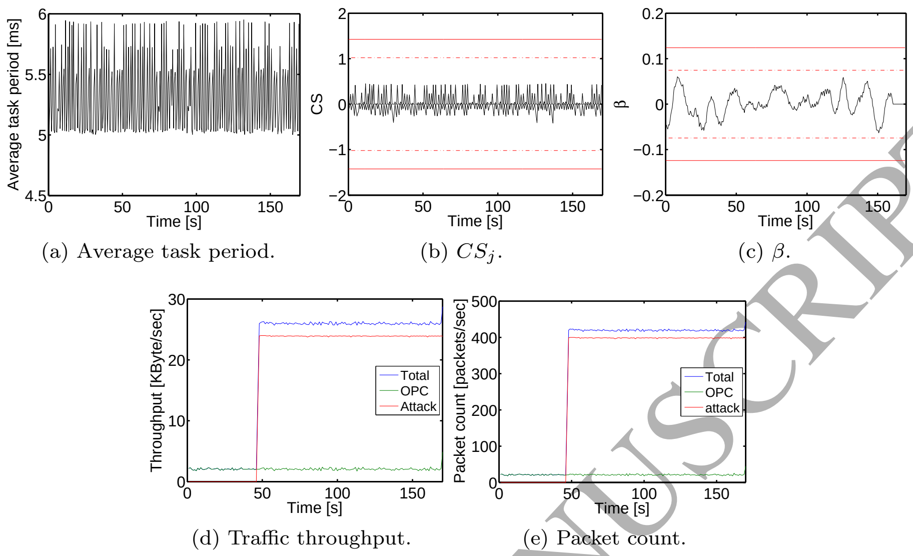


(d) Traffic throughput. (e) Packet count.

Figure 13: Recorded measurements in the case of an `nmap` scan attack against the Siemens S7-1200 controller.

Similarly to the previous studies, we have launched an `nmap` scan (SYN and UDP scans) against the S7-1200 controller. As confirmed by the results in Figure 13, the attack is immediately detected by both the CUSUM-based computation, as well as by the proposed detection algorithm. Nevertheless, the scenario also confirms once again the superior performance of the proposed approach (compared to the traditional CUSUM), since, as demonstrated by the results (see Figure 13c), the algorithm effectively pin-points the start and end of the attack, while the CUSUM-based approach (see Figure 13b) continues to increase even after the attack is halted.

Next, in order to analyze the sensitivity of the S7-1200 controller to network scans, we launched a packet replay attack at a constant rate of 400 packets/sec with the help of the `tcpreplay` tool. In this case, the results were surprising, since, compared to the ILC 350-PN controller, the S7-1200 controller's internal operation (i.e., task scheduling) did not exhibit any notable deviations from the normal. In fact, as demonstrated by the measurements in Figure 14, we did not record any significant deviations. Furthermore, the β values remained



(a) Average task period. (b) CS_j. (c) β.

(d) Traffic throughput. (e) Packet count.

Figure 14: Recorded measurements in the case of a `tcpreplay` attack with 400 packets/sec against the Siemens S7-1200 controller.

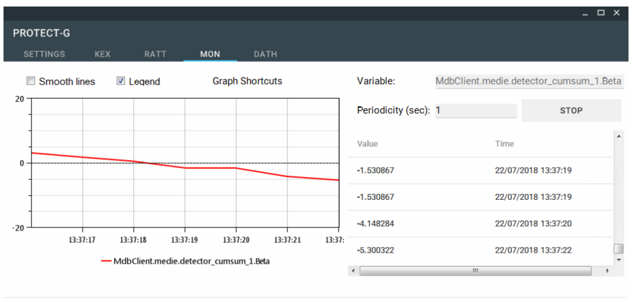
within the limits of the `ALERT` interval during the complete set of measurements. Observe, however, that compared to the attack conducted with the `nmap` tool, the packet rate was significantly lower, while in the former case the packet rate reached a peak value of 3000 packets/sec. Therefore, we can conclude that the proposed algorithm's sensitivity is highly correlated with the controller's sensitivity to external interventions. Accordingly, while the S7-1200 controller is much simpler in design, a higher robustness to external disturbances was observed. As such, the controller is able to schedule at the prescribed fixed rate (with a slight variation between 5ms and 6ms) the preconfigured task period, even in the presence of a significant external disturbance. Nevertheless, we also note that, while the controller is able to monitor the scan cycle and to react if the maximum scan cycle time is exceeded (i.e., by generating a time error), the proposed algorithm can detect the small variations in the expected scan cycle, which otherwise would not trigger any alarms.

6.1. Discussion

The developed detection algorithm brings a paradigm shift in terms of the positioning of security modules within the architecture of industrial installations. The paper advocates that application-layer and software-supported solutions can be integrated into existing control applications without affecting their normal operation. To achieve the safe positioning of anomaly detection algorithms within the application-layer of IC, a careful analysis needs to be performed on the available resources (e.g., programming languages, Application Programming Interfaces – API, computational resources), on the process control logic constraints/restrictions, and on the implementation of the anomaly detection algorithm.

The application-layer positioning of detection algorithms comes with a number of advantages, and, with a few limitations. In terms of advantages, the superiority of the developed methodology stems from its OS and hardware independence. This is owed to the simplicity of the detection algorithm, to the use of simple mathematical computations (additions and multiplications), which are supported by rudimentary controllers (e.g., legacy IC). As further demonstrated by the experimental results performed on two modern and widely used IC (the ILC 350-PN and the S7-1200 controllers), the requirements for integrating anomaly detection into industrial control applications can be safely implemented in IC with notably distinct software architectures. As demonstrated, the algorithm has been developed in two programming languages part of the IEC 61131-3 standard: Structured Text and Ladder Diagrams. Since many IC vendors have decided to make their programming systems compliant with the IEC 61131-3 standard (up to a certain extent), it is therefore reasonable to assume that the proposed approach can be integrated into other controllers as well.

Another notable advantage for leveraging an application-level detection engine is that the approach empowers the IC to become aware of the traffic pattern changes. The alternative would be to position an external detection engine (e.g., external “bump in the wire” solutions [50, 51]). However, this would



Variable: MdbClient.medie.detector_cumsum_1.Beta

Periodicity (sec): 1

Value	Time
-1.530867	22/07/2018 13:37:19
-1.530867	22/07/2018 13:37:19
-4.148284	22/07/2018 13:37:20
-5.300322	22/07/2018 13:37:22

Figure 15: Operation of the detection engine (β values within the graphical user interface developed as part of project PROTECT-G.

require physical intervention and significant changes to the network architecture of industrial installations. These changes would have a dramatic impact on the capital and operating expenses, making the automation solution expensive, highly complex, and difficult to maintain. Conversely, the provisioning of the proposed methodology does not require additional equipment, it does not yield changes to the hardware/network/IC, but most importantly, it has been designed in such a way as to be easily integrated into existing (i.e., legacy) IC. Moreover, the management and operation of the detection algorithm can be performed via traditional industrial protocols (e.g., OPC, Modbus/TCP), which significantly reduces the administrative overhead. An example integration of the developed detection engine into a comprehensive security solution developed as part of project PROTECT-G (Protection of communications in natural gas transportation systems[1]) is depicted in Figure 15. Here, the monitoring of the output of the detection engine (i.e., β values) has been mapped to an OPC variable, which is continuously logged and monitored within a graphical user interface-based application hosted by the Human Machine Interface (HMI).

An important aspect of the developed methodology is that its positioning at the application-level makes it sensitive to any traffic that reaches the IC's network interface. As a result, the approach can record and signal network

[1]http://protect-g.upm.ro/en/index.html

traffic pattern changes not only for the OPC protocol (that is seamlessly integrated with industrial applications), but also for other protocols including serial communications, Modbus/TCP, Modbus/RTU, Fieldbus, etc. As mentioned throughout this paper, the detection algorithm is highly sensitive to the disturbances that ultimately affect the scheduler. Consequently, it can detect network traffic pattern changes (since network packets are handled by interrupt service routines), as well as changes in the program's execution, which can signal a possible abnormal behavior in the control application or in the operation of the physical process (e.g., faults and disruptions).

Lastly, a clear advantage of the proposed paradigm shift is that it is not sensitive to regular operational interventions (e.g., downloading a file, updating the software). Conversely, external detection systems would trigger alarms in case such user-triggered (i.e., legitimate) network traffic would be detected.

In terms of limitations, we note that since the approach is envisioned to be positioned at the application-layer, and it does not directly analyze network traffic, it is therefore susceptible to cyber stealth attacks against industrial systems [48]. However, we note that our methodology does not claim to replace existing anomaly detection techniques (e.g., that may also detect cyber stealth attacks), but it is aimed to empower the IC so that it becomes aware on the changes of network traffic patterns, while providing a cost-efficient solution that can be readily integrated into existing (i.e., legacy) installations. Nonetheless, the approach needs to be seconded by other tools and detection engines that focus on the other dimensions of industrial installations in order to detect process-specific attacks [24, 25, 26, 50], and attacks that cross the boundaries of one dimension (i.e., the cyber-physical attacks) [11, 39, 7].

In terms of future progress and development, we envision to further extend the developed approach with algorithms and external software (running on existing nodes) that aggregate, correlate, and analyze the reports (i.e., β values) of distinct IC in order to identify miss-behaving nodes. Our ultimate goal is to improve the detection rate against cyber stealth attacks. For this purpose, we believe that the alert aggregation and correlation methodology, combined with data fusion techniques [11] would yield a promising approach to detect such advanced attacks. Obviously, a key concern in this respect would be the positioning of these detection algorithms within industrial installations, the protection of communications, and the impact on the overall costs of the developed solution. These are all questions that will be answered as part of future development and improvement of the approach documented in this paper.

7. Conclusions

We developed an innovative methodology for the practical integration of a lightweight anomaly detection algorithm in industrial control applications. The methodology consists of a “monitoring” task and of a detection algorithm that distinguishes itself from previous studies by a simple implementation that can precisely pin-point the attack interval. This represents a salient feature in the designed approach, which can help in the diagnosis procedure of intrusion events. While extensive experimental results on a Phoenix Contact ILC 350-PN controller and on a Siemens SIMATIC S7-1200 controller demonstrated the practical integration of the proposed methodology, we note that the approach does not replace existing security instruments. On the contrary, it aims at providing yet another security tool in the hands of security engineers, in this case, a fully integrated detection engine into critical end-points. Accordingly, our methodology enhances the industrial controller with security features to enable reporting on abnormal network-specific events, which may constitute a significant advantage in the prevention of possibly damaging cyber attacks.

Acknowledgment

This work was supported by a grant of the Romanian National Authority for Scientific Research and Innovation, CNCS/CCCDI-UEFISCDI, project number PN-III-P2-2.1-BG-2016-0013, within PNCDI III.

References

References

[1] 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. Industrial Informatics*, vol. 7, no. 2, pp. 179–186, 2011. [Online]. Available: https://doi.org/10.1109/TII.2010.2099234

[2] S. Stone and M. Temple, "Radio-frequency-based anomaly detection for programmable logic controllers in the critical infrastructure," *International Journal of Critical Infrastructure Protection*, vol. 5, no. 2, pp. 66 – 73, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1874548212000200

[3] A. Almalawi, A. Fahad, Z. Tari, A. Alamri, R. AlGhamdi, and A. Y. Zomaya, "An efficient data-driven clustering technique to detect attacks in scada systems," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 5, pp. 893–906, May 2016.

[4] S. Shitharth and D. P. Winston, "An enhanced optimization based algorithm for intrusion detection in scada network," *Computers & Security*, vol. 70, no. Supplement C, pp. 16 – 26, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167404817300901

[5] J. E. Rubio, C. Alcaraz, R. Roman, and J. Lopez, "Analysis of intrusion detection systems in industrial ecosystems," in *Proceedings of the 14th International Joint Conference on e-Business and Telecommunications (ICETE 2017) - Volume 4: SECRYPT, Madrid, Spain, July 24-26, 2017.*, 2017, pp. 116–128. [Online]. Available: https://doi.org/10.5220/0006426301160128

[6] I. Kiss, B. Genge, P. Haller, and G. Sebestyén, "Data clustering-based anomaly detection in industrial control systems," in *2014 IEEE 10th In-*

ternational Conference on Intelligent Computer Communication and Processing (ICCP), Sept 2014, pp. 275–281.

[7] M. Wan, W. Shang, and P. Zeng, "Double behavior characteristics for one-class classification anomaly detection in networked control systems," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 12, pp. 3011–3023, Dec 2017.

[8] B. Wang and Z. Mao, "One-class classifiers ensemble based anomaly detection scheme for process control systems," *Transactions of the Institute of Measurement and Control*, vol. 0, no. 0, p. 0142331217724508, 2018.

[9] D. Ha, U. Ahmed, H. Pyun, C.-J. Lee, K. H. Baek, and C. Han, "Multi-mode operation of principal component analysis with k-nearest neighbor algorithm to monitor compressors for liquefied natural gas mixed refrigerant processes," *Computers & Chemical Engineering*, vol. 106, pp. 96 – 105, 2017, eSCAPE-26. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0098135417302466

[10] I. Portnoy, K. Melendez, H. Pinzon, and M. Sanjuan, "An improved weighted recursive pca algorithm for adaptive fault detection," *Control Engineering Practice*, vol. 50, pp. 69 – 83, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0967066116300326

[11] B. Genge, C. Siaterlis, and G. Karopoulos, "Data fusion-base anomay detection in networked critical infrastructures," in *2013 43rd Annual IEEE/IFIP Conference on Dependable Systems and Networks Workshop (DSN-W)*, June 2013, pp. 1–8.

[12] A. Di Pietro, S. Panzieri, and A. Gasparri, "Situational awareness using distributed data fusion with evidence discounting," in *Critical Infrastructure Protection IX*, M. Rice and S. Shenoi, Eds. Cham: Springer International Publishing, 2015, pp. 281–296.

[13] B. Chen, D. W. C. Ho, W.-A. Zhang, and L. Yu, "Distributed dimensionality reduction fusion estimation for cyber-physical systems under dos attacks," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. PP, no. 99, pp. 1–14, 2017.

[14] US-CERT, "Alert (TA18-074A): Russian Government Cyber Activity Targeting Energy and Other Critical Infrastructure Sectors," 2018, [Online; Accessed March 2018]. [Online]. Available: https://www.us-cert.gov/ncas/alerts/TA18-074A

[15] T. Yadav and A. M. Rao, "Technical aspects of cyber kill chain," in *Security in Computing and Communications*, J. H. Abawajy, S. Mukherjea, S. M. Thampi, and A. Ruiz-Martínez, Eds. Cham: Springer International Publishing, 2015, pp. 438–452.

[16] Siemens, "Information regarding the Behaviour of SIMATIC S7-1200 in Industrial Networks," 2011, [Online; Accessed March 2018]. [Online]. Available: https://support.industry.siemens.com/cs/document/50428932/information-regarding-the-behaviour-of-simatic-s7-1200-in-industrial-networks?dti=0&lc=en-GB

[17] C. Alcaraz, C. Fernandez-Gago, and J. Lopez, "An early warning system based on reputation for energy control systems," *IEEE Transactions on Smart Grid*, vol. 2, no. 4, pp. 827–834, Dec 2011.

[18] J. Wan, M. K. Khan, M. Qiu, and D. Zhang, "Cloud-assisted industrial systems and applications," *Mobile Networks and Applications*, vol. 21, no. 5, pp. 822–824, 2016.

[19] M. Hagerott, "Stuxnet and the vital role of critical infrastructure operators and engineers," *International Journal of Critical Infrastructure Protection*, vol. 7, no. 4, pp. 244 – 246, 2014.

[20] Symantec, "Dragonfly: Cyberespionage attacks against energy suppliers," *Symantec Security Response*, 2014.

[21] R. Filippini and A. Silva, "A modeling framework for the resilience analysis of networked systems-of-systems based on functional dependencies," *Reliability Engineering & System Safety*, vol. 125, pp. 82–91, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0951832013002676

[22] A. Giani, R. Bent, and F. Pan, "Phasor measurement unit selection for unobservable electric power data integrity attack detection," *International Journal of Critical Infrastructure Protection*, vol. 7, no. 3, pp. 155 – 164, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1874548214000407

[23] J. E. Rubio, C. Alcaraz, R. Roman, and J. Lopez, "Analysis of intrusion detection systems in industrial ecosystems," in *Proceedings of the 14th International Joint Conference on e-Business and Telecommunications (ICETE 2017) - Volume 4: SECRYPT, Madrid, Spain, July 24-26, 2017.*, 2017, pp. 116–128. [Online]. Available: https://doi.org/10.5220/0006426301160128

[24] A. A. Cárdenas, S. Amin, Z.-S. Lin, Y.-L. Huang, C.-Y. Huang, and S. Sastry, "Attacks against process control systems: Risk assessment, detection, and response," in *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, ser. ASIACCS '11. New York, NY, USA: ACM, 2011, pp. 355–366. [Online]. Available: http://doi.acm.org/10.1145/1966913.1966959

[25] J. Giraldo, A. Cardenas, and N. Quijano, "Integrity attacks on real-time pricing in smart grids: Impact and countermeasures," *IEEE Trans. Smart Grid*, vol. 8, no. 5, pp. 2249–2257, 2017. [Online]. Available: https://doi.org/10.1109/TSG.2016.2521339

[26] I. N. 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.

[27] J. Zhao, G. Zhang, and R. A. Jabr, “Robust detection of cyber attacks on state estimators using phasor measurements,” *IEEE Transactions on Power Systems*, vol. 32, no. 3, pp. 2468–2470, May 2017.

[28] Y. Shoukry, P. Nuzzo, A. Puggelli, A. L. Sangiovanni-Vincentelli, S. A. Seshia, and P. Tabuada, “Secure state estimation for cyber-physical systems under sensor attacks: A satisfiability modulo theory approach,” *IEEE Transactions on Automatic Control*, vol. 62, no. 10, pp. 4917–4932, Oct 2017.

[29] E. Pleijsier, “Towards anomaly detection in scada networks using connection patterns,” in *18th Twente Student Conference on IT*, 2013.

[30] R. Barbosa, R. Sadre, and A. Pras, “Towards periodicity based anomaly detection in SCADA networks,” in *17th IEEE Conference on Emerging Technologies & Factory Automation (ETFA 2012)*, 2012, pp. 1–4.

[31] R. R. R. Barbosa, R. Sadre, and A. Pras, “Flow whitelisting in scada networks,” *International Journal of Critical Infrastructure Protection*, vol. 6, no. 3, pp. 150 – 158, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1874548213000437

[32] ——, “Exploiting traffic periodicity in industrial control networks,” *International Journal of Critical Infrastructure Protection*, vol. 13, pp. 52 – 62, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1874548216300221

[33] I. Garitano, C. Siaterlis, B. Genge, R. Uribeetxeberria, and U. Zurutuza, “A method to construct network traffic models for process control systems,” in *17th IEEE Conference on Emerging Technologies & Factory Automation (ETFA 2012)*, 2012, pp. 1–8.

[34] B. Genge, D. A. Rusu, and P. Haller, "A connection pattern-based approach to detect network traffic anomalies in critical infrastructures," in *Proceedings of the Seventh European Workshop on System Security*, ser. EuroSec '14. New York, NY, USA: ACM, 2014, pp. 1:1–1:6. [Online]. Available: http://doi.acm.org/10.1145/2592791.2592792

[35] G. Bernieri, F. Pascucci, and J. Lopez, "Network anomaly detection in critical infrastructure based on mininet network simulator," in *Proceedings of the First Italian Conference on Cybersecurity (ITASEC17), Venice, Italy, January 17-20, 2017.*, 2017, pp. 116–125. [Online]. Available: http://ceur-ws.org/Vol-1816/paper-12.pdf

[36] M. Cheminod, L. Durante, L. Seno, and A. Valenzano, "Detection of attacks based on known vulnerabilities in industrial networked systems," *Journal of Information Security and Applications*, vol. 34, pp. 153 – 165, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2214212616301259

[37] Mitre, "Common Vulnerabilities and Exposures," 2018, [Online; Accessed March 2018]. [Online]. Available: https://cve.mitre.org/

[38] C. Markman, A. Wool, and A. A. Cardenas, "A new burst-dfa model for scada anomaly detection," in *Proceedings of the 2017 Workshop on Cyber-Physical Systems Security and PrivaCy*, ser. CPS '17. New York, NY, USA: ACM, 2017, pp. 1–12. [Online]. Available: http://doi.acm.org/10.1145/3140241.3140245

[39] M. Caselli, E. Zambon, and F. Kargl, "Sequence-aware intrusion detection in industrial control systems," in *Proceedings of the 1st ACM Workshop on Cyber-Physical System Security*, ser. CPSS '15. New York, NY, USA: ACM, 2015, pp. 13–24. [Online]. Available: http://doi.acm.org/10.1145/2732198.2732200

[40] Phoenix Contact GmbH Co. K, "PC WORX 6 IEC 61131-Programming," 2010.

[41] PLCopen Technical Committee 1, TC1, "IEC 61131-3 Programming Languages," 2013.

[42] E. S. Page, "Continuous inspection schemes," *Biometrika*, vol. 41, no. 1/2, pp. 100–115, 1954.

[43] M. Basseville and I. V. Nikiforov, *Detection of Abrupt Changes: Theory and Application.* Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1993.

[44] G. A. Barnard, "Control charts and stochastic processes," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 239–271, 1959.

[45] J. Nadler and N. B. Robbins, "Some characteristics of page's two-sided procedure for detecting a change in a location parameter," *The Annals of Mathematical Statistics*, pp. 538–551, 1971.

[46] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the Association for Computing Machinery*, vol. 20, no. 1, pp. 46–61, 1973.

[47] D. C. Montgomery, *Introduction to statistical quality control.* John Wiley & Sons (New York), 2013.

[48] L. Cazorla, C. Alcaraz, and J. Lopez, "Cyber stealth attacks in critical information infrastructures," *IEEE Systems Journal*, vol. PP, no. 99, pp. 1–15, 2017.

[49] W. J. Conover, "Practical nonparametric statistics," *New York: John Wiley & Sons*, p. 295301, 1971.

[50] A. Khalili, A. Sami, A. Khozaei, and S. Pouresmaeeli, "Sids: State-based intrusion detection for stage-based cyber physical systems," *International Journal of Critical Infrastructure Protection*, vol. 22, pp. 113 – 124, 2018.

[51] X. Jie, H. Wang, M. Fei, D. Du, Q. Sun, and T. Yang, "Anomaly behavior detection and reliability assessment of control

systems based on association rules," *International Journal of Critical Infrastructure Protection*, vol. 22, pp. 90 – 99, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S187454821730046X