Engineering Edge Security in Industrial Control Systems

Piroska Haller, Béla Genge, and Adrian-Vasile Duka

Abstract Industrial Controllers (e.g., Programmable Logical Controllers – PLCs, and Remote Terminal Units – RTUs) have been specialized to deliver robust control strategies. However, little has been done towards the integration of security strategies within their application-layer. This chapter investigates the integration of security solutions within the industrial control system's "edge" devices – the Industrial Controller (IC). As a specific case study it demonstrates the implementation of a simple anomaly detection engine traditional in control applications. The approach shows that the scheduling rate of control applications is significantly affected by various events, such as a change in the number of network packets, configuration interventions, etc. Implementations realized on a Phoenix Contact ILC 350-PN controller demonstrate the feasibility and applicability of the proposed methodology.

1 Introduction

Supervisory Control and Data Acquisition (SCADA) systems provide essential functions for the operation of industrial processes (e.g., oil and gas pipelines, water distribution systems, smart energy systems, air and gas transportation systems). Their architecture combines traditional Information and Communication Technology (ICT) hardware and software with industrial controllers (e.g., Programmable Logical Controllers – PLCs, and Remote Terminal Units – RTUs) in order to facilitate a cost-effective implementation of local actuation strategies.

Industrial controllers (IC) are, in most cases, embedded devices that are 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

P. Haller (✉) · B. Genge · A.-V. Duka
Petru Maior University of Tîrgu Mureş, Tîrgu Mureş, Mureş, Romania
e-mail: phaller@upm.ro; bela.genge@ing.upm.ro; adrian.duka@ing.upm.ro

D. Gritzalis et al. (eds.), *Critical Infrastructure Security and Resilience*,
Advanced Sciences and Technologies for Security Applications,
https://doi.org/10.1007/978-3-030-00024-0_10

have been specialized to deliver robust and effective control strategies, little has been done towards the integration of security strategies within the application of industrial controllers. 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). However, security operations, if carefully engineered, may be integrated into the edge devices found in Industrial Control Systems (ICS), namely, critical devices such as PLCs.

To this end, a considerable amount of research has been focused on the development of Intrusion Detection Systems (IDS) [1, 3, 21–23] for the industrial realm. Different strategies have been developed by leveraging diverse techniques such as classification [16, 25, 26], multivariate statistical analysis including principal component analysis [13, 20], and data fusion [5, 6, 10]. 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 of 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 chapter investigates the integration of security solutions within the ICS's "edge" devices – the IC. As a specific case study it demonstrates the implementation of a simple anomaly detection engine traditional in control applications. The approach leverages the scheduling rates found in control applications, and their deviation from the "normal" behavior. As it turns out, the approach is well-suited for scenarios in which the encountered behavior is sufficiently narrow to allow meaningful detection from the "normal". Furthermore, we show that the scheduling rate of control applications is significantly affected by various events, including a change in the number of network packets, administrator's interventions, configuration changes, etc. Implementations realized on a Phoenix Contact ILC 350-PN controller demonstrate the feasibility and applicability of the proposed methodology.

2 Related Studies

The dramatic impact of traditional computer system attacks on industrial processes has been first demonstrated by the Stuxnet malware [14, 24]. In light of such threats, there have been several proposals towards enhancing the security of existing and future industrial installations [7, 11]. In fact, a considerable amount of research has been allocated to understanding the design of comprehensive anomaly detection systems for industrial systems. Nonetheless, the practical integration of such

schemes was not sufficiently explored. In the remaining of this section we provide an overview of the main anomaly detection techniques for SCADA systems available in the literature.

We start by mentioning techniques that leverage different parameters exposed by the cyber and/or the physical dimension of SCADA systems. The work of Kiss et al. [16], explored the applicability of data clustering techniques for anomaly detection. The Gaussian mixture model was compared to the K-means clustering, and the superior performance of the former was demonstrated. The work of Wan et al. [25], 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. A similar attempt for the classification of different events was undertaken by Wang and Mao in [26]. Here, an ensemble of two models of one-class classification were developed, and the performance of the approach was demonstrated in the context of two industrial installations and several public datasets.

In the direction of multivariate statistical analysis we find the work of Daegeun Ha et al. [13]. Here, the multi-mode principal component analysis (PCA) was used together with k-nearest neighbor algorithm for process monitoring and data classification. The approach was demonstrated in the context of a mixed refrigeration physical process. In a similar direction, a weighted adaptive recursive PCA approach was developed by Portnoy et al. in [20], while the works mentioned in [5, 6, 10] adopted the data fusion strategy as an attempt to develop a multivariate reasoning methodology.

Starting with their work in [4], Cárdenas et al. demonstrated that, by incorporating knowledge of the physical process in the control system, it is possible to detect traditional computer attacks that change the behavior of the underlying control system. 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 [12], Giraldo et al. stepped further and designed an attack-resilient controller. In terms of detection, [12] adopted the non-parametric cumulative sum model.

Lastly, for completeness, we mention that besides the aforementioned techniques, a wide variety of other strategies have been designed for abnormal behavior in industrial settings [3, 8, 9, 15]. Nevertheless, our analysis focused on the techniques leveraging network and process-specific parameters, that may be integrated in the application of industrial controllers.

According to the analysis above 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, the complexity of detection schemes based on computation-intensive operations such as the ones proposed in [5, 6, 10, 13, 16, 20, 25, 26] may require external hardware. Conversely, the integration of process-specific information in the design of the control system, as proposed in [4] and [12], 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 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.

2.1 Architecture of Industrial Controllers

The architecture of modern industrial controllers varies among 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 [19]. Irrespective on the underlying solution, control code is usually organized in several distinct *user* tasks. These are scheduled for periodic execution by the real-time preemptive scheduler (or the real-time operating system), which governs the temporal determinism of user tasks.

Besides user tasks, ICs also host *system* tasks that implement the typical functions for handling remote connections, processing of network packets. 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 can 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 to “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 tasks depends on the priority of each task.

The time between two consecutive runs of the same periodic task (difference between the start time) differs from the task period even for the highest priority task. This is mainly owed to the hardware interrupts and to the execution time of the interrupt service routines. The start 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 illustrated in Fig. 1. Here, we depicted both circular and event-based tasks, together with the monitoring task (i.e. our

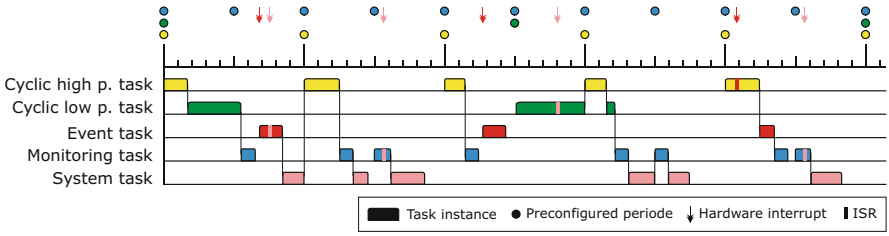

Fig. 1 Example of task scheduling

proposal) and the system task. It can be seen that 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 for ICs.

2.2 Design Considerations

The design of an anomaly detection system for industrial controllers needs to account for several restrictions. While apparently the hardware of modern ICs 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, which significantly reduce the size and complexity of the applicable instruction set.

According to these observations, it is clear that any practical proposal for a detection system needs to be positioned in the user task and it must account for the resources available at this application layer. Therefore, in order to ensure its suitability for existing control applications, we identify two fundamental requirements: (i) the detection algorithm's 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 the user tasks is not affected.

2.3 Developed Anomaly Detection System

The architecture of the proposed anomaly detection system addresses the 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 important requirement is to configure the cyclic monitoring task to skip execution cycles if time overrun occurs, due to the preemption caused by higher priority tasks. An example execution of the proposed monitoring task has been included as part of Fig. 1. 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 (e.g., internal software execution state changes, external cyber attacks), 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.

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_i be the i-th start time of the monitoring task, and $T_i = t_i - t_{i-1}$ the i-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.

Note that the PLC runs a set of periodic tasks, which are repeated after each *hyper-period*. The *hyper-period* is the least common multiple of all task periods, and is large enough such that all tasks are run at least once. Using a moving average filter on the monitoring task, we measure its average period over the hyper-period (over L

consecutive samples), in order to reduce the jitter of different tasks scheduled across different monitoring intervals. In the following, the filtered measured period $\widetilde{T}_j$ is used in the anomaly detection algorithm, computed as follows:

$$\widetilde{T}_j = \sum_{i=(j-1)L}^{jL} T_i/L. \tag{1}$$

In order to minimize the computational overhead and to ensure the proposal's practical integration with existing (legacy) controllers, the proposed detection algorithm is the statistical cumulative sum. A similar technique has been widely used by previous studies in the construction of efficient detection strategies [12]. Briefly, based on the work firstly proposed by Page [18], and Montgomery [17], a two-sided cumulative sum is computed in order to detect the increase and decrease of the mean value of the monitoring task period. The upper (C_j^+) and lower (C_j^-) cumulative sums are computed based on the mean shift value (K), the measured task period (denoted by $\widetilde{T}_j$), the expected task period (denoted by $\hat{T}$), and the deviation of the measured task period σ_T, according to the following two equations:

$$C_j^+ = max\Big(0, C_{j-1}^+ + \widetilde{T}_j - (\hat{T} + K)\Big), \tag{2}$$

$$C_j^- = max\Big(0, C_j^- + (\hat{T} - K) - \widetilde{T}_j\Big), \tag{3}$$

where K is usually expressed through the standard deviation unit $K = \frac{k}{2}\sigma_T$.

According to this detection strategy, a change point is detected when $(C_j^+ \geq h) \cup (C_j^- \geq h)$, where h is the detection threshold. Obviously, the selection of k and h has a major impact on the detection sensitivity and accuracy. Several threshold values can be chosen (e.g., $h_1, h_2, \ldots$) in order to define different levels of criticality, which would also permit to define a different set of actions for each distinct level. Consequently, several distinct detection levels can be chosen to send notifications, stop noncritical system tasks, and to send critical alarm status, reducing thus the negative effects of false alarms to the industrial process.

In case multiple change points need to be detected (i.e., in case a disturbance is persistent), in the above-mentioned method the next starting point is configured as the cumulative sum of the previously measured change point. In this case, the new expected value needs to be computed once again. According to [17], this is computed as follows:

$$\hat{T}^1 = \begin{cases} \hat{T} + K + C_j^+/N^+, & \text{for}(C_j^+ \geq h), \\ \hat{T} - K - C_j^-/N^-, & \text{for}(C_j^- \geq h), \end{cases} \tag{4}$$

where N^+, and N^- denote the number of consecutive periods in which $C_j^- \neq 0$ and $C_j^+ \neq 0$.

3 Experimental Assessment

3.1 Test Infrastructure

The implementation has been tested in the context of a real SCADA system operating in a Romanian gas transportation network. 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 which include the secondary controllers (PLC^S), the Modbus RTU slaves, HMIs, which are all typically found in the automation solution of a gas distribution node in the Romanian gas transportation network. A simplified view of the automation system's architecture is shown in Fig. 2.

Given the significance of the primary controller, the following tests focus on this controller. Each task running on PLC^P is described as a tuple of (period, deadline, [best case worst case execution time], priority). Accordingly, PLC^P runs the following periodic tasks: a control task (100 ms, 200 ms, [10 30], 1); one task for reading inputs (50 ms, 100 ms, [1 10], 1); a Modbus/RTU task (10 ms, 100 ms, [1 2], 0); a Modbus/TCP task (100 ms, 500 ms, [1 2], 2), and monitoring task (10, 200, [1 1], 3). The average filter window length (L) has 100 samples, which permits every task to run at least once.

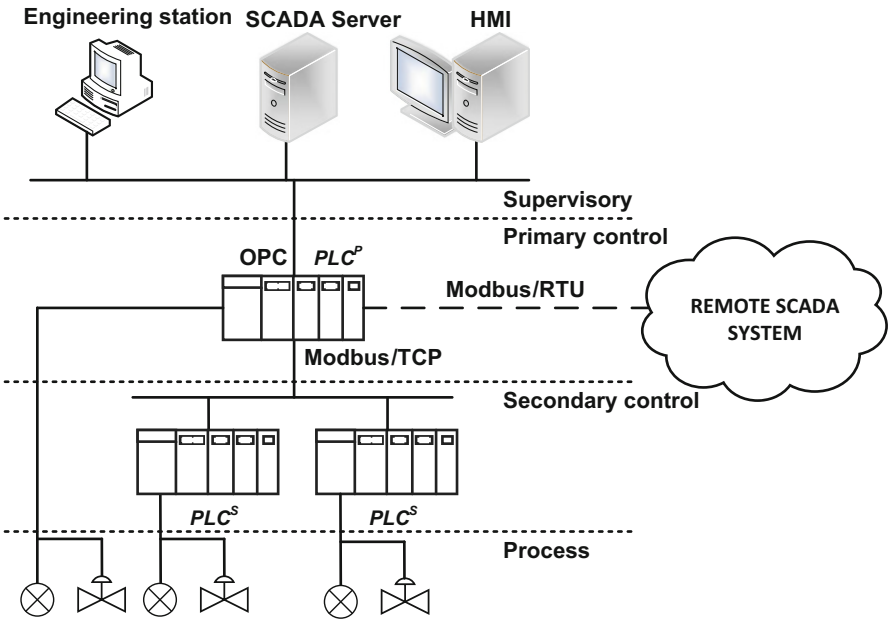

Fig. 2 Simplified architecture of a gas distribution node in a Romanian gas transportation network

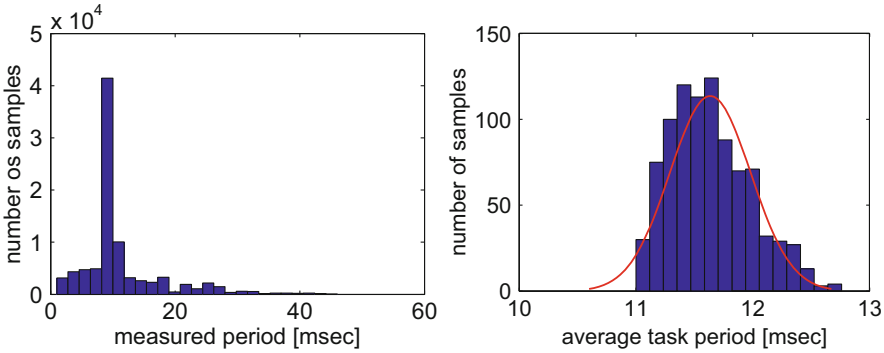

Fig. 3 The distribution of task periods. (**a**) T_i, (**b**) $\widetilde{T}_j$

The implementation and tuning of the proposed detection engine needs to start with an off-line analysis of the recorded attack-free run of the system and in the same time use the existing methods [2] for estimation of the upper bounds on worst case response times based on the task parameters. This is needed to determine the monitoring task period's mean value and deviation. In control systems the asynchronous events have a minimal inter-arrival time, and can be either measured a priori based on the physical process analysis, or can be determined using statistical analysis on the recorded task execution time (Fig. 3a).

Accordingly, the filtered monitoring task period shows a normal distribution (Fig. 3b) with a mean value of 11.63 ms, a standard deviation of 0.346 ms, and a maximum value of 12.76 ms. Even in the disturbance-free run, the expected value of the monitoring task period is larger than 10 ms, since the monitoring task has the lowest priority. The selected parameters for the anomaly detector include half of a deviation shift in both directions ($k = 1/2$), an alert threshold on three standard deviations ($h_1 = 3$), and an error threshold equal to four standard deviations ($h_2 = 4$).

3.2 Measured Results

In order to test the developed anomaly detection system, the network traffic generated by the `nmap` tool (full host scan) was replayed at various packet rates (i.e., the attack traffic) against PLC^P. Figure 4a, b show the behavior and output of the developed anomaly detection system (ADS) in the attack-free scenario. Here, the two threshold levels have been highlighted with horizontal red lines.

Next, in the first set of experiments two 60 s attacks were launched at a 30 s time intervals. The first attack issued 200 packets/s, while the second attack issued 500 packets/s. The detection algorithm was configured to run the two sided

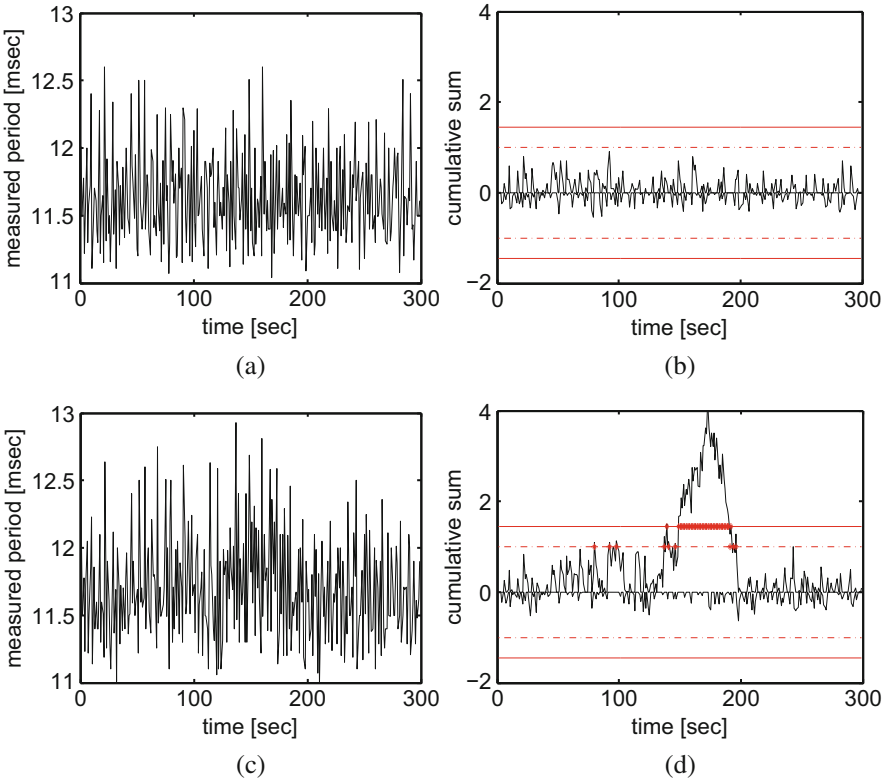

Fig. 4 Monitoring task period and output in the first experimental scenario. (**a**) Task period without attack. (**b**) Detection without attack. (**c**) Task period with attack. (**d**) Detection with attack

cumulative sum. As shown in Fig. 4c, the change in the measured task period is not trivial to notice. However, the anomaly detection algorithm (see Fig. 4d) signals an alert at 100 s, and later on it signals an error level, when the value of the detection algorithm reaches h_2. We observe that the error signal persists even after the attack is stopped (at 170 s). This is a direct consequence of the cumulative sum, which is designed to detect a single point of change. However, if a human intervention is needed when errors are signaled, the identification of the single point of change may not be sufficient.

Next, we launched a second set of experiments with the proposed multiple change point detection algorithm, and with two 120 s attacks launched at 30 s time differences. The first attack replayed the network scan traffic with a 200 packets/s rate, while the second attack increased the replayed packets rate to 700 packets/s. The first rate was chosen to be comparable in the number of packets to the total number of packets processed by the primary controller (PLC^P), while the second rate is equal to the total throughput of PLC^P as shown in Fig. 5.

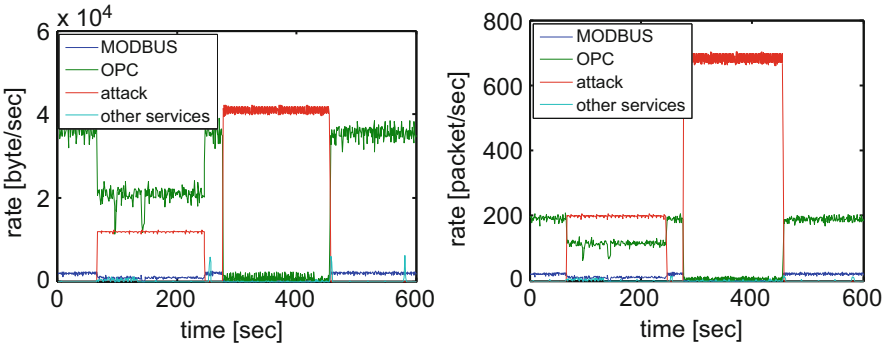

Fig. 5 Network traffic with attack. (**a**) Throughput. (**b**) Packet count

In order to showcase the significance of leveraging multiple change points, two distinct experiment instances have been considered: a first instance that leverages the single change point strategy, and a second instance that leverages the multiple change points strategy. The measured task period for both instances is shown in Fig. 6a.

The results in Fig. 6b show that the magnitude of the attack triggers a significant increase in the value of the cumulative sum. This yields an error state that persists almost up to 600 s, while the attack ended at 450 s. Conversely, in the case of the multiple change point detection strategy, once a threshold is exceeded, the controller estimates the new task period mean (see Fig. 6c) and resets the value of the cumulative sum. Figure 6d shows that the multiple change point detects both the presence and the absence of the attack in a few samples.

The second experiment was repeated under different system load, by changing the control task algorithm including the best case and worst case execution time. The system utilization rate was computed based only on the recorded execution time for the user tasks. The communication tasks and the network traffic have been left unchanged, while the throughput was identical to the one presented in Fig. 5. The mean system utilization rate was chosen between 20% and 50%, but the maximal load did not exceed the schedulability condition. The variation of the system utilization rate is shown in Fig. 7, and the number of detected signals (alert and error) are included in Table 1. As expected, if the system utilization rate is low, the monitoring task scheduling is not delayed by other task or system interrupts generated by the incoming packets. In this case the filtered monitoring task period has a mean value of 10.11 ms, a standard deviation of 0.11 ms for 20% system utilization, a mean value of 12.47 ms, and a standard deviation of 0.527 ms for 50% system utilization. In order to increase the sensitivity on the low utilization rate, the period of the monitoring task can be reduced accordingly.

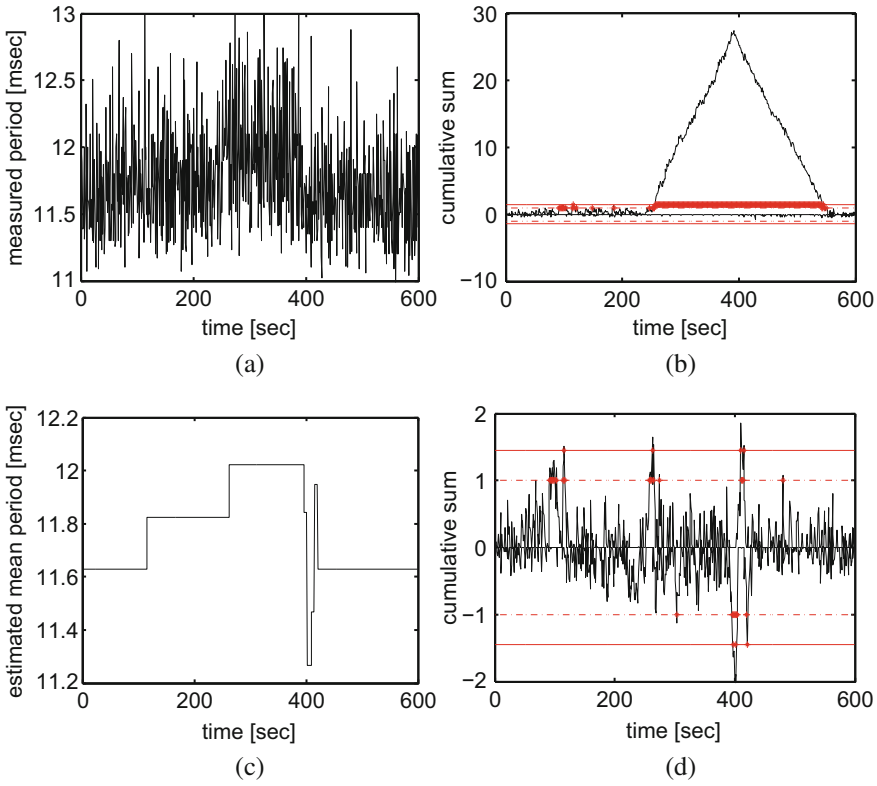

Fig. 6 Monitoring task period and output in the second experimental scenario. (**a**) Task period with attack. (**b**) Detection with single change point. (**c**) Estimated task period mean. (**d**) Detection with multiple change points

Based on the network traffic analysis a significant reduction in OPC network traffic can be observed in the presence of the attack traffic. The throughput on the OPC communication channel was reduced proportionally with the attack traffic and eventually stopped if number of attack packets are increased. Besides this, monitoring only the network traffic is not enough to identify the system anomalies. The third set of experiments modify the OPC traffic rate, by downloading data and code from the engineering station through the OPC communication protocol (see Fig. 8). We observe that when the download/upload operation is initiated, the traffic rate on OPC increases from 40 to 250 kbyte, while the packet rate is adjusted accordingly by the OS running on the PLC. This is needed in order to preserve the packet processing time near a constant value without causing a major impact on

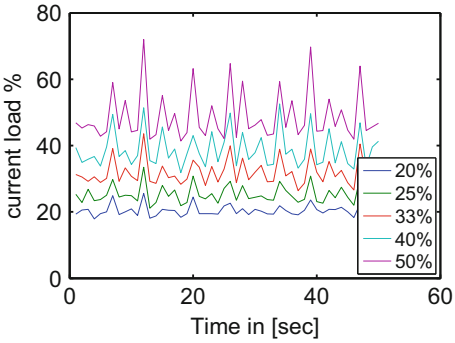

Fig. 7 The variation of the system utilization rate

Table 1 Number of detected alerts for different system utilization rates

	Attack rate	System utilization rate				
Detection level	packet/s	20%	25%	33%	40%	50%
Alert	200	2	3	12	28	164
	700	4	164	179	207	280
Error	200	0	0	4	8	155
	700	1	159	170	200	280

the scheduling of user tasks. As the monitoring task period is not influenced by the regular interventions from the engineering station, the proposed anomaly detection system doesn't generate false alarms.

4 Conclusions

We presented a methodology for enabling edge security in Industrial Control System. In essence, the approach shows that a proper tailoring of operations, together with a careful analysis of industrial controllers (i.e., the edge devices), an effective security module can be integrated into these critical devices as well. More specifically, our proposal is twofold: (i) a monitoring task that repeatedly records the task's start time; and (ii) a lightweight anomaly detection engine based on the computation of the cumulative sum. Intrinsic details have been presented with respect to the functioning of IC, and the implementation of the proposed approach. Experimental results on a real industrial controller confirmed the feasibility and applicability of the approach. As future work, we intend to evaluate its applicability to other industrial controllers as well.

Acknowledgements 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.

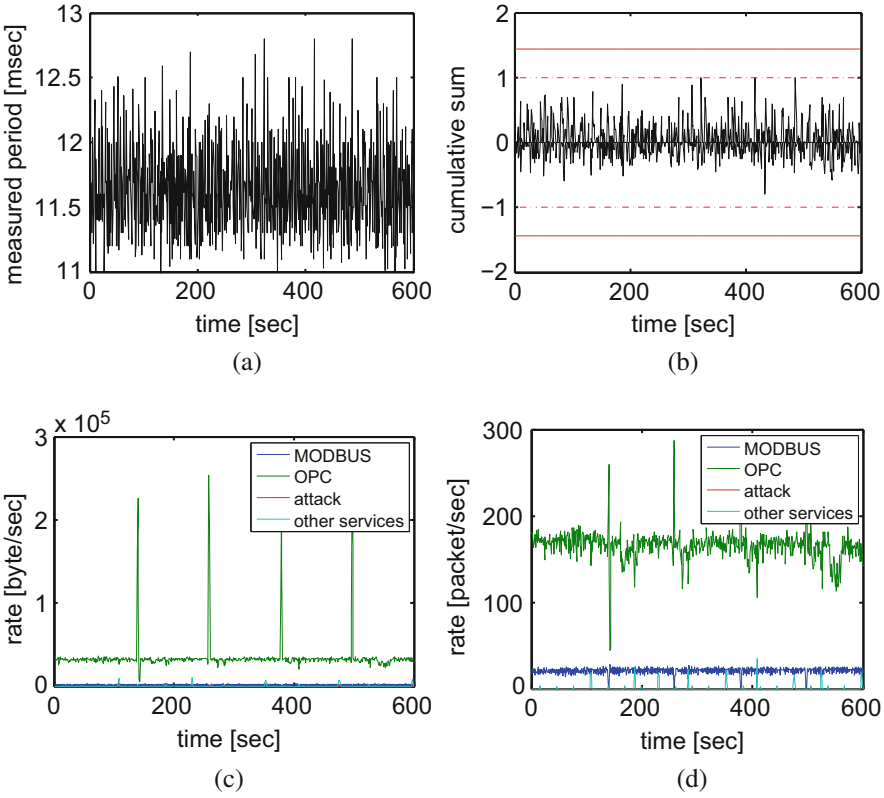

Fig. 8 Monitoring task period, output and network traffic in the third experimental scenario. (**a**) Task period. (**b**) Detection with single change point. (**c**) Throughput. (**d**) Packet count

References

1. Almalawi A, Fahad A, Tari Z, Alamri A, AlGhamdi R, Zomaya AY (2016) An efficient data-driven clustering technique to detect attacks in scada systems. IEEE Trans Inf Forensics Secur 11(5):893–906. https://doi.org/10.1109/TIFS.2015.2512522
2. Bini E, Nguyen THC, Richard P, Baruah SK (2009) A response-time bound in fixed-priority scheduling with arbitrary deadlines. IEEE Trans Comput 58(2):279–286
3. Carcano A, Coletta A, Guglielmi M, Masera M, Fovino IN, Trombetta A (2011) A multidimensional critical state analysis for detecting intrusions in SCADA systems. IEEE Trans Ind Inf 7(2):179–186. https://doi.org/10.1109/TII.2010.2099234
4. Cárdenas AA, Amin S, Lin ZS, Huang YL, Huang CY, Sastry S (2011) Attacks against process control systems: risk assessment, detection, and response. In: Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security, ASIACCS'11. ACM, New York, pp 355–366. https://doi.org/10.1145/1966913.1966959
5. Chen B, Ho DWC, Zhang WA, Yu L (2017) Distributed dimensionality reduction fusion estimation for cyber-physical systems under dos attacks. IEEE Trans Syst Man Cybern Syst PP(99):1–14. https://doi.org/10.1109/TSMC.2017.2697450

6. Di Pietro A, Panzieri S, Gasparri A (2015) Situational awareness using distributed data fusion with evidence discounting. In: Rice M, Shenoi S (eds) Critical infrastructure protection IX. Springer, Cham, pp 281–296
7. Filippini R, Silva A (2014) A modeling framework for the resilience analysis of networked systems-of-systems based on functional dependencies. Reliab Eng Syst Saf 125:82–91. https://doi.org/10.1016/j.ress.2013.09.010, http://www.sciencedirect.com/science/article/pii/S0951832013002676
8. Fovino IN, Coletta A, Carcano A, Masera M (2012) Critical state-based filtering system for securing SCADA network protocols. IEEE Trans Ind Electron 59(10):3943–3950. https://doi.org/10.1109/TIE.2011.2181132
9. Genge B, Rusu DA, Haller P (2014) A connection pattern-based approach to detect network traffic anomalies in critical infrastructures. In: Proceedings of the Seventh European Workshop on System Security, EuroSec'14. ACM, New York, pp 1:1–1:6. https://doi.org/10.1145/2592791.2592792
10. Genge B, Siaterlis C, Karopoulos G (2013) Data fusion-base anomay detection in networked critical infrastructures. In: 2013 43rd Annual IEEE/IFIP Conference on Dependable Systems and Networks Workshop (DSN-W), pp 1–8. https://doi.org/10.1109/DSNW.2013.6615505
11. Giani A, Bent R, Pan F (2014) Phasor measurement unit selection for unobservable electric power data integrity attack detection. Int J Crit Infrastruct Prot 7(3):155–164. https://doi.org/10.1016/j.ijcip.2014.06.001, http://www.sciencedirect.com/science/article/pii/S1874548214000407
12. Giraldo J, Cardenas A, Quijano N (2017) Integrity attacks on real-time pricing in smart grids: impact and countermeasures. IEEE Trans Smart Grid 8(5):2249–2257. https://doi.org/10.1109/TSG.2016.2521339
13. Ha D, Ahmed U, Pyun H, Lee CJ, Baek KH, Han C (2017) Multi-mode operation of principal component analysis with k-nearest neighbor algorithm to monitor compressors for liquefied natural gas mixed refrigerant processes. Comput Chem Eng 106:96–105. https://doi.org/10.1016/j.compchemeng.2017.05.029, http://www.sciencedirect.com/science/article/pii/S0098135417302466. ESCAPE-26
14. Hagerott M (2014) Stuxnet and the vital role of critical infrastructure operators and engineers. Int J Crit Infrastruct Prot 7(4):244–246
15. Haller P, Genge B (2017) Using sensitivity analysis and cross-association for the design of intrusion detection systems in industrial cyber-physical systems. IEEE Access 5:9336–9347. https://doi.org/10.1109/ACCESS.2017.2703906
16. Kiss I, Genge B, Haller P, Sebestyén G (2014) Data clustering-based anomaly detection in industrial control systems. In: 2014 IEEE 10th International Conference on Intelligent Computer Communication and Processing (ICCP), pp 275–281. https://doi.org/10.1109/ICCP.2014.6937009
17. Montgomery DC (2013) Introduction to statistical quality control. Wiley, New York
18. Page ES (1954) Continuous inspection schemes. Biometrika 41(1/2):100–115
19. Phoenix Contact GmbH Co. K (2010) PC WORX 6 IEC 61131-Programming
20. Portnoy I, Melendez K, Pinzon H, Sanjuan M (2016) An improved weighted recursive PCA algorithm for adaptive fault detection. Control Eng Pract 50:69–83. https://doi.org/10.1016/j.conengprac.2016.02.010, http://www.sciencedirect.com/science/article/pii/S0967066116300326
21. Rubio JE, Alcaraz C, Roman R, Lopez J (2017) Analysis of intrusion detection systems in industrial ecosystems. In: Proceedings of the 14th International Joint Conference on E-Business and Telecommunications (ICETE 2017) – vol 4: SECRYPT, Madrid, 24–26 July 2017, pp 116–128. https://doi.org/10.5220/0006426301160128
22. Shitharth S, Prince Winston D (2017) An enhanced optimization based algorithm for intrusion detection in SCADA network. Comput Secur 70(Supplement C):16–26. https://doi.org/10.1016/j.cose.2017.04.012, http://www.sciencedirect.com/science/article/pii/S0167404817300901

23. Stone S, Temple M (2012) Radio-frequency-based anomaly detection for programmable logic controllers in the critical infrastructure. Int J Crit Infrastruct Prot 5(2):66–73. https://doi.org/10.1016/j.ijcip.2012.05.001, http://www.sciencedirect.com/science/article/pii/S1874548212000200
24. Symantec (2014) Dragonfly: cyberespionage attacks against energy suppliers. Symantec Security Response
25. Wan M, Shang W, Zeng P (2017) Double behavior characteristics for one-class classification anomaly detection in networked control systems. IEEE Trans Inf Forensics Secur 12(12):3011–3023. https://doi.org/10.1109/TIFS.2017.2730581
26. Wang B, Mao Z (2018) One-class classifiers ensemble based anomaly detection scheme for process control systems. Trans Inst Meas Control 40(12):3466–3476