CHALMERS
UNIVERSITY OF TECHNOLOGY

The Nuts and Bolts of Deploying Process-Level IDS in Industrial Control Systems

Downloaded from: https://research.chalmers.se, 2019-08-13 09:38 UTC

Citation for the original published paper (version of record):
Almgren, M., Aoudi, W., Gustafsson, R. et al (2018)
The Nuts and Bolts of Deploying Process-Level IDS in Industrial Control Systems
Proceedings of the 4th Annual Industrial Control System Security Workshop : 17-24
http://dx.doi.org/10.1145/3295453.3295456

N.B. When citing this work, cite the original published paper.

research.chalmers.se offers the possibility of retrieving research publications produced at Chalmers University of Technology.
It covers all kind of research output: articles, dissertations, conference papers, reports etc. since 2004.
research.chalmers.se is administrated and maintained by Chalmers Library

(article starts on next page)

The Nuts and Bolts of Deploying Process-Level IDS in Industrial Control Systems

Magnus Almgren
Chalmers University of Technology
Gothenburg, Sweden
magnus.almgren@chalmers.se

Wissam Aoudi
Chalmers University of Technology
Gothenburg, Sweden
wissam.aoudi@chalmers.se

Robert Gustafsson
Chalmers University of Technology
Gothenburg, Sweden
robg@student.chalmers.se

Robin Krahl
University of Freiburg
Freiburg, Germany
krahlr@cs.uni-freiburg.de

Andreas Lindhé
Combitech
Gothenburg, Sweden
andreas.lindhe@combitech.se

ABSTRACT

Much research effort has recently been devoted to securing Industrial Control Systems (ICS) in response to the increasing number of adverse incidents targeting nation-wide critical infrastructures. Leveraging the static and regular nature of the behavior of control systems, various data-driven methods that monitor the process-level network have been proposed as a defensive measure. Although these methods have been evaluated through offline analysis of ICS-related datasets, in absence of documented live experiments in real environments, a complete and global understanding of the applicability and efficiency of process-level monitoring is still lacking.

In this work, we describe our experience of running a fully fledged intrusion detection system in an operational paper factory for 75 days. We discuss the nuts and bolts of running such systems in real environments and underline several practical challenges in meeting ICS-specific requirements. This work essentially aims at bridging the gap between ICS intrusion detection research and practice, and empirically validating the increasingly adopted data-driven approach to process-level monitoring.

KEYWORDS

Intrusion Detection; Industrial Control Systems; PASAD; Cyber-Physical Systems; Deployment; Process-Level Analysis

1 INTRODUCTION

Industrial Control Systems (ICS), often found in critical infrastructures, monitor and operate industrial processes including manufacturing, gas and power distribution, and transportation. The increasing connectivity of ICS to corporate networks and the Internet is rendering these systems vulnerable to cyber attacks capable of causing damage to physical processes, often in safety-critical facilities [1, 6, 15, 16]. Recent studies have shown that, by monitoring the physical process behavior, it is possible to detect sophisticated attacks that have semantic objectives and target the operation of control systems at the physical layer. The key property that enables this approach is the regular dynamics that ICS constantly exhibit owing to the static and cyclic nature of their behavior. Accordingly, various data-driven methods have been proposed in the literature that essentially establish a baseline for the normal behavior based on historical process data and then monitor for deviations attributable to malicious manipulation at the process level [2–4, 8, 19, 20]. These methods have been evaluated through simulations and offline experiments on data extracted from real systems. Simulation platforms (e.g., the Tennessee-Eastman process [5]) and physical testbeds (e.g., the SWaT testbed [9]) allow for crafting attacks and evaluating the detection capabilities of the proposed methods under various attack scenarios. Using datasets extracted from real systems (e.g., pipeline SCADA systems [12] and water treatment plants [8]) adds some degree of realism to the evaluation. *However, in absence of documented live experiments in real environments, a complete and global understanding of the applicability and efficiency of process-level monitoring is still lacking.* In particular, such experiments would shed light on matters pertaining to potential artifacts and peculiarities that may hinder a seamless execution of these methods in practice.

In this work, we consider a Process-Aware Stealthy-Attack Detection mechanism (PASAD) recently proposed by Aoudi et al. [2], build a complete system around it, and deploy a prototype in an operational paper factory. Then, we describe our experience of running the prototype for 75 days, and highlight some technical challenges and practical aspects of live process-level monitoring for intrusions in ICS. Finally, we propose a set of guidelines and recommendations for both security researchers and practitioners who may consider designing or deploying IDS solutions for control systems. The purpose of this work is in large part to bridge the existing simulation-based evaluation efforts with the real world by creating a roadmap characterizing potential hurdles to be expected when bringing the systems into a real environment.

After reviewing related work in Section 2, we discuss the challenges we encountered in Section 3 and present our system design in Section 4. We describe the environments where we tested our system in Section 5 and discuss the lessons learned in Section 6. Finally, we conclude this work in Section 7.

2 RELATED WORK

The research problem of detecting attacks on control systems by monitoring the process-level network using data-driven methods has gained increasing attention over the last few years. The proposed approaches have different characteristics (e.g., whether they are model-based or model-free, the type of process variables they

https://doi.org/10.1145/3295453.3295456

monitor, the techniques they use to learn from historical data, etc). However, they share the common idea that, owing to the static and regular behavior control systems exhibit over time, it is possible to define the normal behavior from historical data, and then detect misbehaviors that are deviants from the norm.

The data-driven methods proposed in the literature are based on various techniques including state-space model identification [20], auto-regression [8], singular spectrum analysis [2], machine learning [7, 12], and data mining [13]. These methods have been evaluated through *(i) simulation* of, e.g., a chemical process (Tennessee-Eastman) [5], electric power flow (MATPOWER) [21], and water distribution piping systems (EPANET) [18]; *(ii) physical testbeds*, including water boilers [8] and the SWaT testbed [9]; and *(iii) offline experiments* on data extracted from real ICS, such as water purification plants [8], water distribution plants [2], and gas pipeline systems [7].

In this paper, we investigate the applicability and deployability of process-level intrusion detection systems adopting the data-driven approach in a *real environment*. We build a complete IDS prototype and run a live experiment in an operational paper factory with the aim of scrutinizing the real-time operation of such systems in practice in order to provide useful guidelines and best practices for both ICS security researchers and practitioners.

3 CHALLENGES

There is a clear difference between testing a process-level IDS solution using a testbed and moving the system to a less controlled real environment. In a testbed setting, it is deceptively easy to run experiments, since the testing environment is controlled, where the system performance is fully observable. Challenges arise when the task is to deploy a stable and resilient prototype in an uncontrolled and undiscovered environment. In this section, we summarize the four main challenges that accompanied the design and implementation of our system.

Challenge A: Unknown Environment

ICS are complex systems involving mainly proprietary software and hardware with models and specifications that vary for each environment. Due to the limited knowledge that IT security researchers are expected to have about the process, data-driven methods that learn the process behavior from historical data are favoured over specification-based methods that attempt to fit a model to the target control system. Unfortunately, however, model-free methods suffer from certain drawbacks. For instance, it is not trivial to decide which sensors to monitor in plants awash with sensors. Furthermore, there is no easy way of knowing what time frame is long enough for training on historical data to capture the sensor signal. As a case in point, the sensor data that we used for training was collected from the real environment over a period of 8 days, which turned out to be too short to capture a whole period of the signal, as we observed during the detection phase of the experiment.

Challenge B: Performance & Footprint

From a security perspective, installing an IDS on a stand-alone device and connecting it to the process-level network is generally a plausible reasoning. However, since ICS control physical processes where attacks can lead to physical damage in a short period of time, it is important that the IDS achieves real-time protection, as well as high memory efficiency for various reasons including, but not limited to, cost, scale of deployment in distributed environments, and deployment on existing ICS components. Hence, the challenge here is to design an IDS with a low memory footprint so that it can be installed on limited-resource hardware, and with fast enough processing capabilities to make real-time decisions.

Challenge C: System Stability

As stated in Challenge B, running the IDS on cheap limited-resource hardware is a favorable approach, however, this comes along with some challenges pertaining to the stability of the system. In particular, the IDS needs to be robust enough to handle sudden crashes, packet loss, and issues that may arise from bad memory management. In this regard, logging mechanisms can help identify the roots of such problems for troubleshooting. However, in order to remove the burden from ICS staff to maintain yet another system, it is desirable to automate the procedure for handling such incidents by achieving system stability and resilience against unexpected failures. For instance, during the design phase, we had to account for the case where the IDS stops receiving sensor values, either due to a system crash or because of dropped packets, by running the detection component as a service and scheduling automatic restarts.

Challenge D: Trust & Data Access

At the preliminary stage, the IDS requires data for training and determining the parameters. Due to the proprietary nature of ICS, it is quite difficult to get process data out of these environments to process in a university lab. Interestingly, it turns out that it is easier to get the prototype into the environment to perform the training without exporting data from the facility. Also, it is worth mentioning that it is not necessarily easy to find people with the knowledge needed to run a network dump or set up a mirror port on a router. Either they are OT technicians with little knowledge about monitoring network traffic and extracting data, or they are IT technicians with limited access to the process. Furthermore, trust issues entailed lack of online access to our prototype, which we had to compensate for by backing up process data and logs to a USB drive, and asking ICS staff to communicate the most recent backup to us every once in a while.

4 SYSTEM DESIGN

Our system consists of two main components: PASAD and Midbro. PASAD has recently been proposed by Aoudi et al. [2] as a process-level attack detection technique that monitors solely raw sensor measurements. As PASAD is specification-agnostic, the choice of this method helps us resolve Challenge A since we are dealing with an unknown environment and have little knowledge about the dynamics of the system we are monitoring.

During real-time operation, PASAD requires sensor data to be served on the fly as soon as they are available. To this end, we

built the Midbro component, which lies at the core of our system. Midbro captures network traffic, extracts relevant process data, and serves them to PASAD through a dynamic buffering mechanism that ensures a timely and reliable delivery.

In the following, we describe PASAD and Midbro, then we discuss the choice of suitable hardware for the prototype.

4.1 PASAD

PASAD [2] is a model-free process-level detection mechanism that continuously monitors sensor measurements in ICS to check if the system operation is drifting from historical normal behavior. The method takes as input an observed time series of sensor readings and works in two phases: an offline learning phase and an online detection phase. Initially, by leveraging the static and regular nature of control systems, the normal behavior is mathematically defined through analysis of historical process data without the need to create a model of the underlying physical process. To detect attacks, PASAD then measures the degree to which present sensor observations conform with the estimated dynamics. The method has been validated using various datasets and has been shown capable of detecting sophisticated stealthy attacks that cause rather subtle structural changes in the sensor signal. In the following, we present a brief overview of PASAD to make this paper self-contained. The reader may refer to [2] for a thorough treatment of the underlying theory, basic steps, and parameters setting.

PASAD uses a time-series analysis technique known as *singular spectrum analysis* to extract signal information representing the deterministic behavior of the control system, purely from noisy sensor measurements. More specifically, a time series of raw sensor measurements $\mathcal{T} = x_1, x_2, \cdots, x_N, x_{N+1}, \cdots$ is initially embedded in a vector space. Afterwards, during a learning phase, a *signal subspace* is identified through a mathematical procedure, onto which *training vectors* are projected. The projected vectors occupy a bounded region in this subspace and thereby form a cluster. During a detection phase, a sliding window composes a new *test vector* at every iteration containing the most recent sensor measurement, and the distance between the test vector and the centroid of the cluster is computed. Under attack conditions, the computed distance is presumed to increase, signaling that the physical process is departing from the normal state. Following is a formal description of the two phases of PASAD.

4.1.1 Learning Phase.

In the learning phase, an initial subseries of $\mathcal{T}$ of length N is unfolded into the column vectors of a *trajectory* matrix $\mathbf{X} = [\mathbf{x}_1 : \mathbf{x}_2 : \cdots : \mathbf{x}_K]$ by forming K L-lagged vectors $\mathbf{x}_i = (x_i, x_{i+1}, \cdots, x_{i+L-1})^T$, where L is called the lag parameter, $1 \leq i \leq K$, and $K = N - L + 1$. The *singular value decomposition* of $\mathbf{X}$ is performed to obtain an orthonormal set of L eigenvectors $\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_L$ of the covariance matrix $\mathbf{X}\mathbf{X}^T$. A matrix $\mathbf{U} = [\mathbf{u}_1 : \mathbf{u}_2 : \cdots : \mathbf{u}_r]$ is then formed, whose columns are the $r < L$ leading eigenvectors, where r is the so-called *statistical dimension*. The training vectors $\mathbf{x}_i, 1 \leq i \leq K$, are then projected onto the signal subspace spanned by the column vectors of $\mathbf{U}$, and the centroid of the cluster they form is computed as $\tilde{\mathbf{c}} = \mathbf{U}^T\mathbf{c}$, where $\mathbf{c}$ is the sample mean of the training vectors.

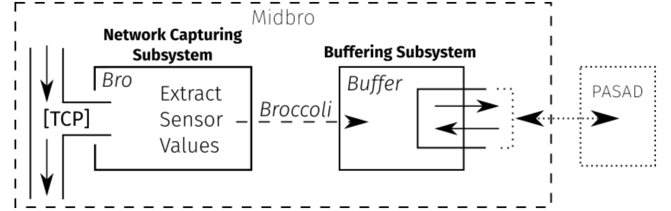

Figure 1: An overview Midbro's main components. Communication with external systems is depicted by dotted shapes.

4.1.2 Detection Phase.

In the detection phase, a *departure score* is computed, at every iteration, for the most recent lagged vector $\mathbf{x}_j, j > K$. This is done by computing the squared Euclidean distance between the test vector $\mathbf{x}_j$ and the centroid $\tilde{\mathbf{c}}$ as $D_j = ||\tilde{\mathbf{c}} - \mathbf{U}^T\mathbf{x}_j||^2$. Finally, an alarm is generated whenever D_j crosses a prespecified threshold. The authors apply what they refer to as the *isometry trick* to speed up computations during the necessarily real-time detection phase. Hence, the detection procedure is quite fast, requiring a single matrix multiplication for every distance computation.

We perform the offline training in MATLAB, using sensor data extracted from the real environment. The output of this procedure that will be used in the detection phase consists of the matrix $\mathbf{U}$, the centroid $\tilde{\mathbf{c}}$, and the free parameters $N, L,$ and r. Based on our C implementation of the detection component of PASAD, the memory footprint turned out to be rather small, where only 160KB of memory is needed on average to store the L most recent sensor values and the variables from the learning phase.

4.2 Midbro

We built Midbro to extend PASAD into a complete and deployable system.[1] It comprises two subsystems: *(i)* a *network capturing engine* that captures and processes network traffic; and *(ii)* a *buffering mechanism* that intermediately stores register values and serves them to PASAD. The system is connected in a pipeline workflow as depicted in Figure 1.

4.2.1 Network Capturing Engine.

The capturing subsystem listens to a network interface and provides a framework for capturing and parsing incoming packets. For a particular register on a particular machine, the packet payloads are filtered out, and corresponding events that hold the extracted register content are created. To capture and parse network traffic, Midbro utilizes the network analysis framework Bro [14]. Bro is built from the ground up to be flexible, providing a versatile and powerful scripting engine that allows for extending the framework to support new protocols. It works on three different layers as illustrated in Figure 2. Initially, the *libpcap* library is responsible for capturing packets from the network interface. The captured packets are then handled by the event engine that parses the packets and generates corresponding events. Finally, these events are handled by policy scripts, which we implemented using the custom Bro scripting language. The policy scripts can access data extracted by the event engine, trigger warnings, write to a file, and even generate new events.

[1] Available at https://github.com/lindhe/midbro

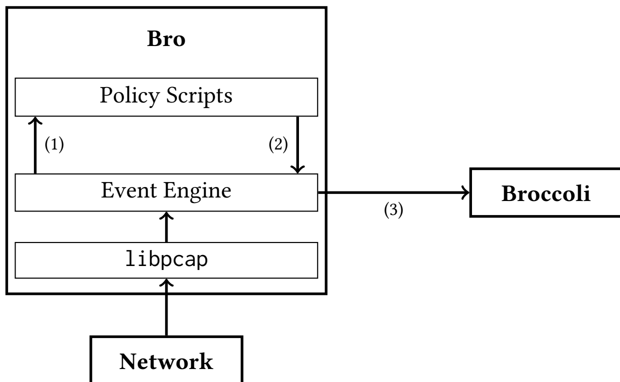

Figure 2: The different layers of Bro. Events generated by the Bro core for the parsed traffic are handled by our policy script (1), which performs the transaction matching and generates new events (2), which in turn are handled by Midbro using the Broccoli library (3).

The machines we monitor in our experiments communicate using the Modbus protocol described in Section 5.1, which is supported by Bro. After the events for requests and responses generated by Bro for the different function calls are handled by the policy script, they are inspected to identify the relevant registers from which the actual sensor values are subsequently extracted.

4.2.2 Buffering Mechanism. The register values that come along with the events are placed in a buffer, where they are held until they are requested by PASAD, which uses function calls to interface with the buffering subsystem. Since Bro's policy-script engine is single-threaded, it is imperative that the most recent register value is released promptly after it has been extracted from a captured Modbus packet. However, as PASAD may be busy with other tasks at any given time, it may not be feasible to collect values directly from the capturing engine. Instead, values are sent via the Bro event engine to a *FIFO buffer*, which in turn serves the values to the IDS as soon as the latter is ready to receive new data.

Evidently, though, one would run into the same problem if the buffer in turn was single-threaded. Therefore, we implemented the buffering engine with two threads: one producer thread and one consumer thread. At one end, the producer thread constantly listens to the socket where Bro events are sent. As soon as a new event arrives at that socket, the sensor value is extracted from the event and then added to the tail of a queue. At the other end, the consumer thread performs the task of popping values from the queue and serving them to PASAD upon request. In order to ensure data integrity, the threads are synchronized using a binary semaphore, which verifies that the same data is not accessed by both threads at the same time. In addition, counting semaphores are used to keep track of the available space on the buffer, blocking the consumer if it tries to pop a value from an empty buffer.

To handle the situation where the buffer becomes full, we chose to implement a head-drop policy, where the oldest value in the buffer is dropped before inserting a new one. This has the advantage of always serving the most recent values to PASAD, rather than causing new values to be delayed (or dropped) in favour of older values in the buffer.

The buffering mechanism is subscribed to the Bro events via *Broccoli*—the Bro client communications library—as shown in Figure 1, which allows for easy interaction between Bro scripts and the native code. The communication between Midbro and PASAD is enabled by a request-response based API, where function calls are used to request a number of new sensor values from the buffer.

4.3 Choice of Hardware

With the software components in place, we needed to decide on a suitable hardware platform for the prototype. Ultimately, taking into account the resource constraints in ICS, a process-level IDS software for control systems should be designed to work on limited-resource hardware, such as programmable logic controllers (PLCs), without exhausting their resources in terms of processing time and memory usage. For our prototype, we aimed at easily accessible and affordable hardware. Broadly speaking, running the IDS on a small single-board computer is very practical, since it makes on-site installation easy and out-of-the-way for the ICS staff.

The recent release of the Raspberry Pi 3+ brought a Gigabit Ethernet and a higher clocked processor than previous generations, thereby making it more suitable for use cases like network analysis, and a good choice for the prototype. The Odroid platform, which also features a high-performance processor and a Gigabit Ethernet NIC, could as well serve the purpose of our design. The more expensive Odroid-UX4 and Odroid-C2 models, which have 2 GB of RAM (compared to 1 GB for the Raspberry Pi) and more cores, were excluded as possible targets since we aimed at maintaining a small memory footprint, and we would hardly benefit from more than 2 cores. The cheapest Odroid-C1+ model is quite similar to the Raspberry Pi 3+ in terms of performance, but much less available and widespread. We therefore settled on the Raspberry Pi, which is widely used and has an active community behind it.

Additionally, the Raspbian distribution is a relatively lightweight and easy-to-deploy OS, and comes pre-packaged with both Bro (`bro`) and the communications library Broccoli (`libbroccoli-dev`) in the official repositories, which is a particularly desirable feature for such experiments.

5 EXPERIMENTS

The central purpose of this work is to explore the challenges of bringing a research IDS prototype into a real environment. Evaluating the detection capability of attacks on ICS is out of the scope of this paper.

Before deployment in the real environment, we wanted to verify that our prototype is stable enough and that corner cases, which might prove costly to treat at a later stage, are addressed beforehand. Therefore, we set up a testbed locally in a controlled environment to run the necessary tests.

In this section, we begin with a description of the prevalent Modbus protocol. Then, we describe both the controlled experiment performed in a university lab and the live experiment performed in the real environment, and discuss performance results. In Section 6, we show that although the controlled experiment was valuable and contributed to a more stable system, the real environment still introduced unforeseen artifacts.

5.1 Modbus Communication

The Modbus network protocol [10] provides read and write access to data in a network. The protocol variant encountered in our experiments is the *Modbus TCP/IP* [11], which is used for communication using the TCP/IP architecture. A node in a Modbus network may provide access to different types of data: coils, discrete inputs, holding registers and input registers, identified by a function code. Coils and discrete inputs, typically used for configuration and calculations, store one bit each, while input and holding registers store 16 bits each, can only be read, and are typically used for sensor measurements. A function call is initiated by a request from one node (client) in the network to another node (server), such that the latter reacts to the request with a response, typically stating the success and the result of the function call. As the IDS monitors sensor measurements, we are only interested in continuous values. Therefore, coils and discrete inputs are disregarded in our design.

When multiple registers need to be accessed by a function, both the address of the first register and the number of registers are specified. However, responses to a read request do not contain the address of the register that was read. Therefore, in order to extract all the necessary data, information about both requests and responses should be used.

Each register is assigned an address between 0 and 65535, which is specific to the server that manages the register. To identify a register on a Modbus node, both the register type and the address are needed. The addressing of nodes in Modbus TCP/IP is done using IP addresses.

Each node in the network can act both as a client and as a server. Also, it is possible for a client to issue several requests to the same server simultaneously without waiting for responses. Requests and responses are matched using a Transaction ID (TID), which is unique to each client-server pair.

5.2 Controlled Experiment: Local Testbed

For the controlled experiment, we set up a local testbed in a university lab. Using a network switch with a port-mirroring feature, we set up a host that replays previously recorded Modbus traffic onto the network.

The data used for this experiment consists of network traffic captured from equipment inside an operational water distribution plant. The 105GB Modbus/TCP traffic capture was recorded over a period of 106 days.

The system where the captured data originates from has a Human-Machine Interface (HMI), which operators use to monitor process variables transmitted from different substations (e.g., pressure or water level in a tank) or to issue commands to substations when necessary to control the industrial process. In total, there are 24 connected hosts in the network, two of which are Modbus masters and the rest are slaves. The control server where the HMI is located is responsible for relaying commands to every slave PLC in the network. In addition, there are three communication interfaces used to send Modbus commands across the network to the slave devices.

One notable distinction between the live environment and this local testbed is that we are in control of the network traffic data rate. As a result, we could induce traffic interruptions by pausing the data replay, in order to reliably emulate the event of traffic disruption (e.g., downtime due to factory maintenance). Furthermore, with the local testbed, we could monitor the network capturing and data analysis in real time. This was a valuable resource that helped us confirm that the system did indeed work as intended, and could handle data interruptions and sudden system restarts.

Another important reason for running a controlled experiment before deployment in the real world is to observe the effect of long-term use. While the core functionality of the system might seem to work properly during shorter tests, there could be issues with the code or system setup that might lead to a crash after a longer time of capturing and data analysis. Therefore, before launching the live experiment, we let our system run for over a weekend in the controlled environment, and encountered no issues.

5.3 Live Experiment: The Paper Factory

We conducted the live experiment in an operational paper factory in Sweden. The main product of the factory is produced from paper pulp, which is fed into the paper production line depicted in Figure 3, and undergoes several procedures until it becomes paper.

We focus on monitoring the sensors responsible for reading the water content of the paper surface. These sensors are placed on an ABB NP1200 control frame, shown in Figure 4, at the end of the production line.

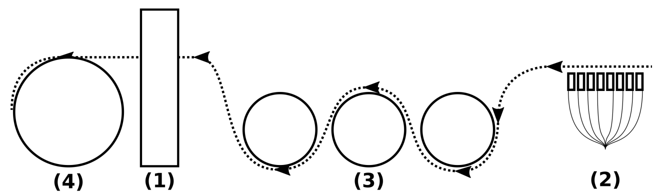

Figure 3: The paper production line, where (1) marks the control frame from which we gather data; (2) marks the dilution water valves controlled according to a feedback loop from the control frame sensors; (3) marks the steam-filled barrels used to dry the paper; and (4) marks the final paper roll.

The control frame communicates the sensor data via an ABB I867 Modbus interface to the ABB PM866 controller. The Modbus interface has network monitoring capabilities, and represents the entry point for us to observe the sensor values. Our system is connected via Ethernet to a switch, which in turn provides a connection to the Modbus interface.

The Modbus TCP/IP packets sent from the control frame hold register data from every new sensor reading since the last transmission. Many sensors report 32-bit float values, which are stored in two 16-bit Modbus registers using the IEEE-754 binary32 format [22]. The stream of packets typically contains hundreds or even thousands of register values.

We deployed our prototype and let it run for 75 days without any intervention, except for extracting data regularly to track the system stability and performance.

5.4 Discussion & Results Overview

We now discuss the results of the live experiment in terms of stability and performance of our system through analysis of data gathered throughout the deployment period.

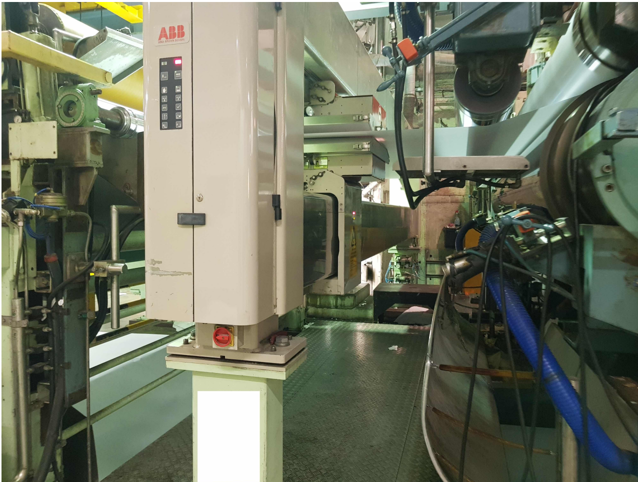

Figure 4: The ABB NP1200 control frame from which we gather sensor readings.

The raw sensor measurements and the corresponding departure scores computed by PASAD are shown in Figure 5. Since this was an attack-free experiment, we expected to see a stable detection behavior, which was indeed the case. Training was performed on sensor measurements spanning a period of 8 days. To determine the alarm threshold, as recommended in [2], we ran the IDS for a validation period on an initial subseries (highlighted in blue) spanning 15 days of normal operation, and then set the threshold to a value that is slightly higher than the maximum distance attained. Based on this selection, as shown in Figure 5, no false alarms were observed during the remaining testing period.

A curious fact that is worth pointing out is the small gaps that can be observed in the sensor readings, indicated by the red vertical bars in Figure 5. In an attempt to investigate this lack of sensor data, we compared the corresponding time intervals with the downtime logs that were provided to us by our contact at the facility, only to arrive at a match and conclude that the data silence was due to downtime in the factory. Interestingly, the interruptions were handled gracefully by PASAD, which did not exhibit any strange behaviors during these periods, a fact that contributed to the stability of our system during real-time operation.

Additionally, we monitored the system performance to determine whether the choice of hardware was adequate and to troubleshoot any potential problems. The system performance measurements shown in Figure 6 correspond to CPU and memory usage. The CPU usage was particularly important to monitor, since packets are likely to be dropped had the usage sustained a maximum activity for long periods. Conveniently, the CPU usage maintained a level of 50% and lower, except for momentary spikes to 100%, indicating the likelihood that there have been no notable packet drops due to insufficient system resources. Furthermore, the memory usage shows a stable behavior throughout the experiment. Note that the sharp drops in allocated memory, specifically on April 26th and May 23rd, match with the more significant downtimes at the factory.

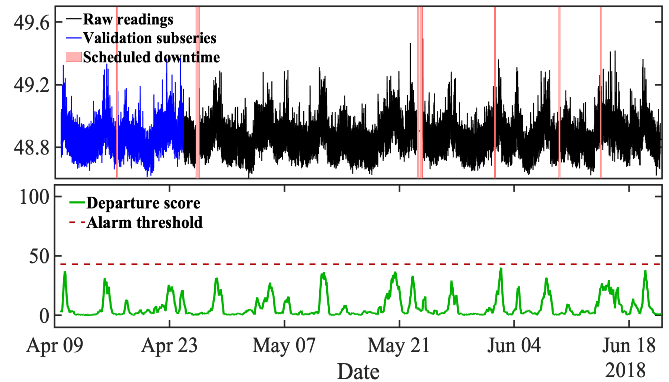

Figure 5: PASAD detection results showing a stable behavior.

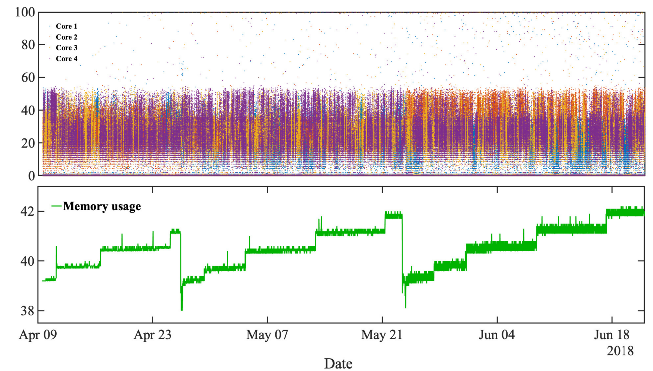

Figure 6: Performance results in terms of CPU and memory usage.

6 LESSONS LEARNED: GUIDELINES & RECOMMENDATIONS

In this section, we express the knowledge we gained from the experience of running a process-level IDS in a real environment in the form of a set of guidelines that we recommend security researchers and practitioners to consider when designing or deploying IDS solutions for ICS.

6.1 Process Knowledge

We learned that, in a real environment, even a process-agnostic IDS is not truly plug-and-play. There are several practical considerations when deploying a prototype in a real ICS that should not be overlooked and which are not compensated for by using a model-free approach. For example, it is likely that multiple machines will communicate over the same network, requiring knowledge from both OT and IT personnel in order to identify relevant machines for monitoring. This is further complicated by the fact that each machine might employ hundreds of sensors. Although we had reasons to suspect this would be the situation already during the controlled experiment with the data from the water plant, it became most evident once we were on site. The control frame we were receiving data from has hundreds of sensors, and each sensor may require more than a single register. Specifically, there were in excess of 1,300 registers from a single machine to keep track of. After discussions with our contact person from the factory, we settled on monitoring a single register that we knew would hold data relevant

to the process. Knowing which registers are of relevance requires in-depth understanding of the OT environment, beyond what can be expected from IT professionals. However, automated data characterization and classification techniques as proposed in [8] may contribute to making this part of the deployment less cumbersome.

6.2 Signal Data Disruptions

There are two points that we wish to emphasize when it comes to (unexpected) disruptions to the analyzed signal. First, it is not intolerable to drop values occasionally. We already knew this was the case for minor interruptions during the controlled experiment, but during the real-world experiment, there was a period of roughly 18 hours, where no data was received due to scheduled maintenance at the factory, and as can been seen in Figure 5, the situation was handled gracefully by PASAD.

Second, it is not trivial to distinguish between a crashed network capturing process and an actual interruption of the process data stream, and that is not a major issue either. Our prototype is built to assume the worst case scenario (i.e., crashed network capturing), in which case it reboots if it detects no new values within the past 30 minutes, in an attempt to recover. To the best of our knowledge, we had at no point a system reboot because of a crashed process; the downtimes were only due to maintenance stops at the factory. A positive consequence of liberally restarting the system is that the need for manual restarts is reduced. During conversations with the factory staff, this seemed to be an important point to consider.

6.3 Modbus Parsing

Bro turned out to be a useful and convenient framework for the task of capturing and extracting Modbus data. It took us only a few weeks to build the necessary scripts for the network capturing subsystem. The initial worry was that Bro might not be fast enough to keep up with the stream of signal data in an industrial application. But that turned out not to be the case in both the controlled experiment and the live experiment, where the data rate was too slow to saturate our system.

One major downside of Bro, however, is that it is single-threaded. Therefore, implementing advanced data analysis techniques directly using Bro scripts might not be advisable. Instead, we suggest having an external solution to do the analysis, as we did with the buffering mechanism. Maintaining as little work as possible in Bro is a fairly good strategy, since Bro needs to get back to handling new incoming data packets as soon as possible. We have shown that a pretty simple buffering engine goes a long way toward overcoming performance issues caused by the fact that Bro is single-threaded. Not only does a buffer help with bursts of traffic, but it also provides an elegant way of interacting with other processes, especially on hardware with several cores, without the need to block the Bro thread.

An alternative to Bro could have been Snort [17], which is a prominent network analysis tool. Compared to Bro, Snort lays more emphasis on intrusion *detection and prevention*, rather than being a general framework. But since it is open source, it could most likely be extended to perform most of the tasks we used Bro for. One of the main reasons we favoured Bro over Snort is that we found it easier to develop the code we needed in Bro scripting language.

Finally, one noteworthy caveat is that Bro does not match corresponding requests and responses for the Modbus protocol. To work around this, the transaction ID specified in the Modbus headers can be inspected. Also, as it is possible for one client to send multiple requests without waiting for the responses, it is not sufficient to only store the last request in a connection. Instead, a table of requests indexed by their transaction IDs in the connection object should be maintained.

6.4 Buffering

Using a buffering mechanism proved quite beneficial, because even when the average data rate is slower than what the IDS can handle, new values might appear in bursts, which can temporarily outpace the IDS. A sufficiently large buffer is likely to mitigate this effect by allowing the IDS to work asynchronously when handling the sensor readings. Thus, by having new values added to the tail of the queue, the rate of incoming packets would trump the rate of the IDS, ensuring that the most recent value available is served.

Whenever the consumer thread can not keep up with the data rate on the network, the buffer will fill up. Once the buffer is full, unless PASAD (or any other IDS for that matter) starts to consume at least at the same rate as new values are produced, the buffer will need to drop values. Since real-time performance is of essence, we consider that adopting the head-drop policy as mentioned in Section 4.2.2, where the oldest packets are dropped, is advantageous because it would ensure that the consumer can always be served the most up-to-date data.

7 CONCLUSION

Although research on intrusion detection in the context of classical corporate IT networks goes back a long way, the fundamentally different nature of ICS has led researchers to take more viable approaches by leveraging distinctive ICS properties, such as their tendency to have static topologies and regular traffic at the process level. Several data-driven methods that capture the normal behavior of the physical process from historical data and then monitor for deviations therefrom have been proposed and investigated. However, the evaluation of these methods seems to have been restricted to simulations and offline analysis of relevant datasets. In this work, we take the evaluation of process-level monitoring a step further by running a fully fledged prototype in a real environment to examine the feasibility of the proposed methods in real-world settings. We reported lessons learned from a live experiment in an operational paper factory lasting 75 days and proposed a set of guidelines and best practices for both security researchers and practitioners who may consider deploying such solutions in real environments.

ACKNOWLEDGMENTS

The research leading to these results has been supported by the Swedish Civil Contingencies Agency (MSB) through the project "RICS" and by the European Community's Horizon 2020 Framework Programme through the UNITED-GRID project under grant agreement 773717. At the time of writing this article, all authors were affiliated with the Department of Computer Science and Engineering at Chalmers University and Gothenburg University.

REFERENCES

[1] Marshall Abrams and Joe Weiss. 2008. Malicious Control System Cyber Security Attack Case Study—Maroochy Water Services, Australia. *McLean, VA: The MITRE Corporation* (2008).

[2] Wissam Aoudi, Mikel Iturbe, and Magnus Almgren. 2018. Truth Will Out: Departure-Based Process-Level Detection of Stealthy Attacks on Control Systems. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18)*. ACM, New York, NY, USA. https://doi.org/10.1145/3243734.3243781

[3] Alvaro Cardenas, Saurabh Amin, Bruno Sinopoli, Annarita Giani, Adrian Perrig, Shankar Sastry, et al. 2009. Challenges for Securing Cyber Physical Systems. In *Workshop on Future Directions in Cyber-Physical Systems Security*, Vol. 5.

[4] Alvaro A. Cárdenas, Saurabh Amin, Zong-Syun Lin, Yu-Lun Huang, Chi-Yen Huang, and Shankar Sastry. 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, NY, USA, 355–366. https://doi.org/10.1145/1966913.1966959

[5] James Downs and Ernest Vogel. 1993. A Plant-Wide Industrial Process Control Problem. *Computers & Chemical Engineering* 17 (1993), 245–255.

[6] Nicolas Falliere, Liam O Murchu, and Eric Chien. 2011. W32. Stuxnet Dossier. *White paper, Symantec Corp., Security Response* 5, 6 (2011), 29. https://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf

[7] Cheng Feng, Tingting Li, and Deeph Chana. 2017. Multi-level Anomaly Detection in Industrial Control Systems via Package Signatures and LSTM Networks. In *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 261–272. https://doi.org/10.1109/DSN.2017.34

[8] Dina Hadžiosmanović, Robin Sommer, Emmanuele Zambon, and Pieter H. Hartel. 2014. Through the Eye of the PLC: Semantic Security Monitoring for Industrial Processes. In *Proceedings of the 30th Annual Computer Security Applications Conference (ACSAC '14)*. ACM, New York, NY, USA, 126–135. https://doi.org/10.1145/2664243.2664277

[9] Aditya Mathur and Nils Ole Tippenhauer. 2016. SWaT: A Water Treatment Testbed for Research and Training on ICS Security. In *Cyber-physical Systems for Smart Water Networks (CySWater), 2016 International Workshop on*. IEEE, 31–36. https://doi.org/10.1109/CySWater.2016.7469060

[10] Modbus Protocol 2012. MODBUS Application Protocol Specification V1.1b3. http://modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf

[11] Modbus Variant 2006. MODBUS Messaging on TCP/IP Implementation Guide V1.0b. http://modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf

[12] Patric Nader, Paul Honeine, and Pierre Beauseroy. 2014. Lp-Norms in One-Class Classification for Intrusion Detection in SCADA Systems. *IEEE Trans. Industrial Informatics* 10, 4 (2014), 2308–2317.

[13] Shengyi Pan, Thomas Morris, and Uttam Adhikari. 2015. Developing a Hybrid Intrusion Detection System Using Data Mining for Power Systems. *IEEE Transactions on Smart Grid* 6, 6 (Nov 2015), 3104–3113. https://doi.org/10.1109/TSG.2015.2409775

[14] Vern Paxson. 1999. Bro: A System for Detecting Network Intruders in Real-Time. *Computer networks* 31, 23-24 (1999), 2435–2463.

[15] Lee Robert, Michael Assante, and Tim Conway. 2014. German Steel Mill Cyber Attack. *SANS Industrial Control Systems* 30 (2014), 62. https://ics.sans.org/media/ICS-CPPE-case-Study-2-German-Steelworks_Facility.pdf

[16] Lee Robert, Michael Assante, and Tim Conway. 2016. Analysis of the Cyber Attack on the Ukrainian Power Grid. *Electricity Information Sharing and Analysis Center & SANS Industrial Control Systems* (March 2016). https://ics.sans.org/media/E-ISAC_SANS_Ukraine_DUC_5.pdf

[17] Martin Roesch. 1999. Snort - Lightweight Intrusion Detection for Networks. In *Proceedings of the 13th USENIX Conference on System Administration (LISA '99)*. USENIX Association, Berkeley, CA, USA, 229–238. http://dl.acm.org/citation.cfm?id=1039834.1039864

[18] Lewis Rossman. 1999. The EPANET Programmer's Toolkit for Analysis of Water Distribution Systems. In *WRPMD'99: Preparing for the 21st Century*. 1–10. https://doi.org/10.1061/40430(1999)39

[19] David I. Urbina, Jairo A. Giraldo, Alvaro A. Cardenas, Nils Ole Tippenhauer, Junia Valente, Mustafa Faisal, Justin Ruths, Richard Candell, and Henrik Sandberg. 2016. Limiting the Impact of Stealthy Attacks on Industrial Control Systems. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16)*. ACM, New York, NY, USA, 1092–1105. https://doi.org/10.1145/2976749.2978388

[20] David I Urbina, David I Urbina, Jairo Giraldo, Alvaro A Cardenas, Junia Valente, Mustafa Faisal, Nils Ole Tippenhauer, Justin Ruths, Richard Candell, and Henrik Sandberg. 2016. *Survey and New Directions for Physics-Based Attack Detection in Control Systems*. US Department of Commerce, National Institute of Standards and Technology.

[21] Ray D Zimmerman, Carlos E Murillo-Sánchez, and Deqiang Gan. 1997. MATPOWER: A MATLAB Power System Simulation Package. *Manual, Power Systems Engineering Research Center, Ithaca NY* 1 (1997).

[22] Dan Zuras, Mike Cowlishaw, Alex Aiken, Matthew Applegate, David Bailey, Steve Bass, Dileep Bhandarkar, Mahesh Bhat, David Bindel, Sylvie Boldo, et al. 2008. IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2008* (2008), 1–70.