Through the Eye of the PLC: Semantic Security Monitoring for Industrial Processes

Dina Hadžiosmanović
Delft University of Technology
The Netherlands

Robin Sommer
ICSI/LBNL
Berkeley, CA
USA

Emmanuele Zambon
University of Twente
SecurityMatters B.V.
The Netherlands

Pieter H.Hartel
University of Twente
The Netherlands

ABSTRACT

Off-the-shelf intrusion detection systems prove an ill fit for protecting industrial control systems, as they do not take their process semantics into account. Specifically, current systems fail to detect recent process control attacks that manifest as unauthorized changes to the configuration of a plant's programmable logic controllers (PLCs). In this work we present a detector that continuously tracks updates to corresponding process variables to then derive variable-specific prediction models as the basis for assessing future activity. Taking a specification-agnostic approach, we passively monitor plant activity by extracting variable updates from the devices' network communication. We evaluate the capabilities of our detection approach with traffic recorded at two operational water treatment plants serving a total of about one million people in two urban areas. We show that the proposed approach can detect direct attacks on process control, and we further explore its potential to identify more sophisticated indirect attacks on field device measurements as well.

1. INTRODUCTION

Industrial control systems (ICS) monitor and control physical processes, often inside critical infrastructures like power plants and power grids; water, oil and gas distribution systems; and production systems for food, cars, ships and other products. As these environments differ from traditional IT systems, they also face unique security challenges that render effective protection challenging. Off-the-shelf intrusion detection systems (IDS) prove a particularly ill fit. Classic signature matching requires precise patterns of anticipated intrusions—an unrealistic assumption in a setting where attacks remain rare overall, yet may carefully target their victims—and existing behavioural approaches fail to incorporate the domain-specific context of operating in these specialized environments [18]. While a few network IDS now include ICS-specific protocol support, their capabilities remain limited to finding low-level technical attacks, such as protocol violations and buffer overflow exploits. They fail, however, to address the fundamentally different threat of malicious changes to a plant's *process control*. Stuxnet represents the most prominent example of such an attack: it manipulated the speed of centrifuges to run outside of their operational range, eventually causing physical damage. While Stuxnet demonstrates a level of sophistication that is feasible only for the most resourceful attackers, SABOT [26] lowers the bar for similar attacks by recovering the mapping between the memory of programmable logic controllers (PLC) and the actual process inputs.

To report process control modifications, an IDS can monitor the status of PLC variables: these attacks manifest as unauthorized changes to a device's parametrization (e.g., setpoints, status). In practice, however, it proves challenging to provide an otherwise independent IDS with access to such process-level PLC activity. The combination of a PLC's embedded nature, its critical role in a plant's operation, and its often proprietary internals, all but prohibits creating a direct interface to an external monitor. It turns out, however, that an alternative approach can provide the relevant information without touching the device: as any configuration changes must reach the PLC through the ICS' network infrastructure, one can passively monitor the network traffic for the corresponding update commands. Doing so proves non-obtrusive and, hence, easy to deploy and scale: it requires just a standard network tap that provides the IDS with a copy of the PLCs' traffic.

In this work we develop such a semantic, network-based IDS. Our system extracts process operations from a PLC's raw network packets, and then constructs a corresponding time series for each process variable to understand its expected activity. We derive variable-specific forecasting models that partially borrow from the process safety community's established domain expertise, and we assess their predictive power with an initial objective to understand where they work well, yet also cases that fail to yield a stable baseline. For our analysis we use Modbus traffic recorded over two-week periods at two operational water treatment plants serving a total of about one million people in two urban areas.

Our results show that this approach allows for straightforward detection of *direct* attacks on process control: they manifest as changes to variables that normally would stay stable, enabling to detect the corresponding attacks that the recent literature discusses. We further explore the potential for extending this approach to *indirect* process control attacks, which reflect only as deviations in field measurements,

ACSAC '14 December 08 - 12 2014, New Orleans, LA, USA

ACM 978-1-4503-3005-3/14/12 ...$15.00.
http://dx.doi.org/10.1145/2664243.2664277.

either because of tampering with sensors or due to a direct control change through an unobservable channel. At first, we find that here the real world introduces artefacts that limit the power of our initial models. However, when we investigate a series of illuminating cases in more detail, we identify possible improvements that can significantly increase their prediction capabilities.

Overall, we consider this work as a step towards non-obtrusive, semantic security monitoring for ICS networks. To the best of our knowledge, no prior work has yet examined operational industrial processes to a similar level of detail from a network vantage point. We structure the remainder of this paper as follows. In §2 we discuss process control attacks in more detail. In §3 we present our monitoring approach, along with a testbed scenario (§3.2) that we use for demonstrating the capabilities of the presented techniques. §4 summarizes our implementation, and §5 evaluates our approach. In §6 we present related work, and §7 provides concluding remarks.

2. DETECTING PROCESS ATTACKS

A typical ICS includes a number of common components, including a human-machine interface (HMI), supervisory infrastructure, field devices, and communication infrastructure. For our work we focus on programmable logic controllers (PLCs), which constitute embedded devices that run custom code to control the field devices (e.g., regulate the speed of a pump). For an attacker, the PLCs provide the most effective point to penetrate, as they control the target processes. To better understand the spectrum of possible network-based attacks on PLCs, we surveyed plausible scenarios carried out over the Modbus TCP protocol as a case study. In Table 1 we aggregate the attacks into three categories. Basic *Level 1* attacks operate at the IP or TCP level, such as manipulating packet sizes. *Level 2* attacks violate semantics of the Modbus protocol at the payload-level, such as breaking the protocol conventions or transmitting values outside the range of either the protocol specification or what the receiving side supports (e.g., invalid function codes). Finally, *Level 3* attacks represent activity that is legal at the protocol level, yet violates semantic constraints that a process imposes, including both semantically incorrect messages (e.g., conflicting commands) and operations that lead the site into an undesirable state (e.g., a command to open a pump when it must remain shut). We distinguish three subtypes of such *process* attacks: *(i)* reconnaissance (e.g., discovering process configuration), *(ii)* direct control (e.g., modifying process setpoints) and *(iii)* indirect control (e.g., tampering with exchanged variable values to trigger undesirable reaction). *Level 3* represents the most challenging target for detection, and three recent examples of (primarily) *direct control* attacks demonstrate the fundamental limitations of current systems to report them. First, Carcano et al. [10] present a proof-of concept malware that manipulates values be generating Modbus packets carrying syntactically correct commands, yet with inverted values. Second, SABOT [26] performs a semi-automated analysis of PLC code and high-level process descriptions to identify key variables for then crafting malicious attacks that divert a process; in contrast to the work of Carcano, this attack assumes a highly knowledgeable attacker. Finally, Stuxnet constitutes the most prominent real-life attack on process integrity. By attacking PLCs, it crossed a boundary as the first publicly known malware that injected semantically meaningful commands into a highly specific plant environment. Stuxnet managed to divert a process by generating malicious yet technically valid control commands that changed the behaviour of centrifuges. Additionally, Stuxnet had an indirect component tampering with the flow of information presented to the operators, causing them to miss the attack's progress.

All three examples manifest as illegitimate write operations that update process settings to harmful values. However, monitoring PLCs for such changes externally proves challenging as hardly any site will allow an IDS to interface with the devices directly. In this work we pursue an indirect, passive way of tracking PLC activity by recreating process semantics from their network communication. In the following we show that doing so enables inferring process behaviour at a level sufficient to detect the three direct attacks we discuss above. We further demonstrate the potential of network-level semantic monitoring for finding indirect process control attacks as well. Our approach does generally not depend on the attacker's knowledge about the plant. For direct attacks we assume that the logic for issuing PLC updates remains uncompromised, whereas for indirect attacks we do not impose any constraints.

3. APPROACH

We now present our approach to detect attacks that aim to manipulate the process control, both direct and indirect. It proceeds in three main phases: *(i) extraction* distills current variable values out of network traffic; *(ii) characterization* divides the observed process variables into three categories that we examine separately; and *(iii) modelling and detection* derives behavioural models for each variable and reports when new observations deviate from what they predict. We discuss these phases individually in §3.3–3.5, after first introducing technical background in §3.1 as well as a small testbed setup in §3.2 that we use for illustration.

3.1 Background

Conceptually, we find two semantic groups of network communication between ICS components: *(i)* process awareness, and *(ii)* process control. *Awareness* propagates status information about the controlled process across devices (also referred as *annunciators* in [26]). In particular, the ICS supervisory infrastructure requests regular updates from the PLCs to report the current plant status to its operators. In addition to escalating critical updates for timely reaction, awareness also collects trending data for long-term process analysis. A typical update cycle spans a few seconds. PLCs also propagate awareness information across themselves to ensure that each device learns sufficient information about critical variables before entering the next process stage (e.g., PLC 1 might require information about the state of a field device connected to PLC 2 before starting a subsequent process stage). *Process control* is generally exercised in one of two ways: *(i)* by PLCs according to their embedded logic; and *(ii)* by operator commands that override PLC internal logic. In either case it is the PLC that carries out the action, and hence will reflect the process change as an update to its internal state.

PLC's process variables characterise the current operation state. Examples of typical variables include the setpoint (i.e., configuration setting) for a physical process, the cur-

Table 1: Summary of plausible attacks against PLC implementations: Modbus example

Level	Impact		Attack description	Example
1	Data integrity		Corrupt integrity by adding data to the packet.	Craft a packet that has a different length than defined in parameters or in spec [2].
2	IT System	Reconnaissance	Analyse functionality a PLC implements.	Probe FC, listen for responses and exceptions [2].
		Integrity	Exploit lack of specification compliance.	Manipulate application parameters within spec (e.g., offset) or outside of spec (e.g., illegal FC) [2, 9, 37].
			Perform unauthorized use of an administrative command.	Use FC 8-0A to clear counters and diagnostics audit [2].
		Denial of service	Perform MITM to enforce system delay.	Send exception codes 05, 06 or FC 8-04 to enforce Listen mode [2].
			Perform unauthorized use of administrative command.	Use FC 8-01 to restart TCP communication [2, 9].
3	Process	Reconnaissance	Analyse structure of memory map.	Probe readable/writable points. Exceptions tell process implementation details [2].
		Direct control	Perform change on process variable.	Write inverted or min/max values [10]. Modify key setpoint variables [14, 26].
		Indirect control	Tamper with process values.	Replay values [14].

FC: Function code defining the type of functionality in Modbus. *MITM: Man-in-the-middle attack.*

rent value of a valve sensor, and the current position in a cycle of program steps. Process variables serve as input to the PLC code. For example, a variable value representing a high pressure level might trigger the start of a draining stage. Likewise, the PLC carries out operator commands by writing into corresponding variables. For example, a command to open a valve would update a variable that the program code is regularly checking; once it notices the update, it outputs the corresponding analogue signal to the physical device.

Within the device, process variables map to PLC memory cells through the data model of the PLC. The data model determines the representation of the process data both inside the PLC's memory and on the network level, usually tying in directly with the communication protocol that a vendor opts to use. The actual mapping is determined by a PLC programmer and thus remains specific to each PLC instance. Examples of data models in different protocols include: Modbus using *memory maps* of 16-bit *registers* and 1-bit *coils*; Profinet defining *slots*, *subslots*, and *channels*; and MMS using *objects* to address process variables [21].

3.2 Testbed Scenario

In §5 we validate our approach using network traffic from water purification operational plants as our work-load. However, to illustrate capabilities in a controlled environment, we here set up a small testbed consisting of one PLC and one HMI workstation, using a simplified version of a demonstration kit from a corresponding ICS vendor.

The process under control comprises six plant components (see Figure 1): a tank, a heater, two valves, a level sensor, and a temperature sensor. The process consists of

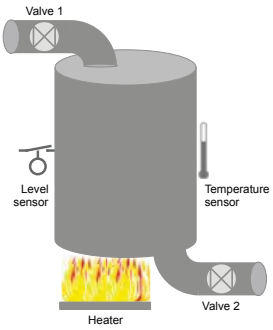

Figure 1: Process setup of the testbed environment

three operations that repeat continuously: tank filling, water heating, and tank draining. The HMI collects nine variables for measurements (tank level and water temperature), control (e.g., valve status, tank level setpoint), and reporting (tank level alarm). Table 2 shows the relevant parts of the PLC's memory map.

As a safety constraint, we implement an alarming system that triggers when process variables reach undesirable values. To illustrate the detection, we use two attack scenarios: *(i)* changing the level setpoint to overflow the tank; and *(ii)* sending the PLC tampered measurement information to trigger process changes or mislead operator awareness. These attacks closely relate to the surveyed attacks from the literature: the first resembles *direct process attacks* (e.g., SABOT and Stuxnet-like attacks) while the second corresponds to *indirect process attacks* (e.g., measurement tampering by Stuxnet).

Table 2: Testbed PLC memory map

Reg.	Name	Type	Desc
HR0010	V1On	bool	Status of valve 1
HR0011	V2On	bool	Status of valve 2
HR0012	HeaterOn	bool	Status of the heater
HR0020	TankLevelSP	fixpoint	SP tank level (L)
HR0021	TankLevel	fixpoint	Level of the tank (L)
HR0022	TempSP	fixpoint	SP water temp.
HR0023	Temp	fixpoint	Water temp (celsius)
HR0030	TankLevelAl	enum	Alarms tank level
HR0031	TempAl	enum	Alarms water temp.

SP: setpoint

3.3 Data Extraction

The data extraction phase is a preprocessing step that distills the values of process variables out of ICS network traffic. It consists of two subparts: *(i)* parsing the application-layer network protocol to extract the relevant commands, including all their parameters; and *(ii)* constructing *shadow memory maps* inside the analysis system that track the current state of all observed process variables, providing us with a mirror of the PLCs' internal memory.

For this work we focus on parsing Modbus, in which each command comes with a set of parameters as well as a data section. Basic parameters include function/sub-function codes that define the operation, an address reference specifying a memory location to operate on, and a word size giving the number of memory cells affected. The data section includes the actual values transmitted, i.e., the current value for a *read* operation and the intended update value for a *write*. We maintain shadow memory maps by interpreting each exchanged message according to its semantics, updating our current understanding of a PLC's variables accordingly.

The following (simplified) commands from our testbed setup (see §3.2) illustrate the extraction step.

```
Time 1: PLC 1, UID: 255, read variable 21, value: 10
Time 2: PLC 1, UID: 255, read variable 21, value: 14
Time 3: PLC 1, UID: 255, read variable 21, value: 18
```

After processing the messages, the shadow memory map will report 18 as the current value for variable 21. As we know from the testbed's configuration, that variable corresponds to the *tank level* and, hence, will reflect three distinct types of behaviour: an increasing trend during filling, a constant value during heating, and a decreasing trend during the draining phase. Indeed, our extraction step confirms this expectation: The following list represents a small excerpt from variable 21's values, as extracted from actual network traffic inside the testbed:

```
...490,492,494,496,498,500,500,500,500,500,500,
...496,492,488,484,480,476,472,468,464,460,456, ...
```

We have confirmed that these extracted values indeed match what the PLC stores internally over time.

3.4 Data Characterisation

Next, we perform a *characterization* phase that separates PLC variables into different categories. Based on interviews with plant engineers, we identify four general groups: *(i) control*: variables for configuring plant operation (e.g., device setpoints, configuration matrix); *(ii) reporting*: variables for reporting alarms and events to operators through HMI or other PLCs (e.g., pump load is too high); *(iii) measurement*: variables reflecting readings from field devices and sensors (e.g., current tank level, current water flow), *(iv) program state*: variables holding internal PLC state such as program counters, clocks, and timeouts. While the character of variables varies with their groups, we observe three cases that suggest specific models for predicting future behaviour: most variables either *(i)* change continuously, and gradually, over time; *(ii)* reflect attribute data that draws from a discrete set of possible values; or *(iii)* never change. The first is typical for sensor measurements; program state and reporting tend to use the second; and the third proves common for process settings (e.g., setpoints). Taking a specification-agnostic approach, there is no definite resource to directly tell what type of data a variable reflects—recall from §3.1 that memory maps are specific to each PLC instance. Thus, we apply heuristics to categorize process variables according to the behaviour we observe. In our testbed we can directly cross-check if the results indeed match the configuration.

More specifically, our heuristics leverage an observation—corroborated by plant engineers—that reporting variables are typically encoded in bitmaps that, depending on the number of distinct reporting events, map to discrete set of values. We use this as criteria for differentiating between the three categories by counting the number of distinct values we see for each variable across a training set. If we observe only a single value, we consider the variable to represent a constant value. We consider a set of up to 2^k distinct values, for a chosen k, as an attribute series. If we see more than 2^k values for a variable, we assume it holds a continuous range.

In our testbed environment, we set the parameter $k = 3$. Running the characterisation with 2 hrs of its network traffic leads to classifying 4 of the 9 variables as constant, 3 as attributes, and 2 as a continuous—which aligns correctly with their actual semantics.

3.5 Data Modelling and Detection

Now that we can distinguish between constant, attribute and continuous time series, we proceed with building behavioural models.

Modelling. To model constant and attribute data, we derive a set of expected values (i.e., the enumeration set for attribute data; the one observed value for constant series). To model continuous data, we leverage two complementary techniques, *autoregression modelling* and *control limits*. Since we do not tailor our model to any specific process description, we choose to leverage techniques that are *(i)* relevant to the context (i.e., commonly used in safety monitoring) and *(ii)* do not pose strong assumptions on the monitored time-series. Autoregressive modelling represents a common technique to capture the behaviour of a correlated series, such as successive observations of an industrial process [38]. An autoregressive model of order p states that x_i is the linear function of the previous p values of the series plus a prediction error term [8]:

$$x_i = \phi_0 + \phi_1 x_{i-1} + \phi_2 x_{i-2} + ... + \phi_p x_{i-p} + \epsilon_i$$

where $\phi_1, ..., \phi_p$ are suitably determined coefficients and ϵ_i is a normally distributed error term with zero mean and non zero variance σ^2. Using the autoregressive model, we can make one-step estimations for future values of the process variable underlying the time series. This way, the model can be used to detect stream deviations. However, as for any regression, a set of small changes can take the stream

outside of its operational limits without exhibiting regression deviations [38]. To address this, we apply a complementary strategy, namely Shewart control limits [38]. The control limits represent a pair of values $\{L_{min}, L_{max}\}$ that define the upper and lower operational limit of the process variable. Typically, the limits are calculated as values that are three standard deviations from the estimated mean.

While the presented techniques are generally deployed in time series analysis, including specifically in the safety domain, our application approach is *(i)* broader, applied indiscriminately on the full spectrum of exchanged messages (i.e., data series) and *(ii)* more generic, posing no assumption on the expected behaviour of the analysed data series. While necessarily exhibiting a lower accuracy than approaches would that tailor specifically to the semantics of individual variables, our modelling enables building a comprehensive network monitor for a wide range of processes.

Detection. For constant and attribute series we raise an alert if a value in a series reaches outside of the enumeration set. To detect deviations in continuous series, we raise an alert if the value *(i)* reaches outside of the control limits or *(ii)* produces a deviation in the prediction of the autoregressive model. More specifically, for estimating the deviation in the autoregressive model, we compare the residual variance (observed during training) with the prediction error variance (observed during testing). A prediction error variance that is significantly higher than the residual variance implies that the real stream has deviated from the estimated model, and thus we should raise an alert [19].

To illustrate the detection capabilities in a controlled environment, we test our approach on two semantic attacks crafted for the process operating in our testbed. The first attack effectively consists of a command that changes the tank level setpoint (HR0020 in Figure 2). As a result, the tank's filling phase (HR0021 in Figure 2) continues until the water level overflows the tank capacity. Results show that this attack is detected as *(i)* a deviation in the setpoint variable and *(ii)* a value reaching the maximal control limit $Lmax$. The second attack consists of a set of commands that tamper with the value for the temperature level (HR0023 in Figure 3). As a result, the tank's filling phase terminates early, the heating process begins but immediately stops, and then the draining process starts. As a consequence, both the heater and the boiler may get damaged. In this second scenario no alarm is generated by the PLC (since the observed levels never reach any undesirable value), yet our approach detects a deviation in the behaviour.

Our results demonstrate the expected capabilities (and limitations) of both approaches [38]. In particular, an autoregressive model is effective for detecting sudden changes (e.g., detection of the second attack) while control limits are generally a good strategy for maintaining the specific mean level of a process variable (thus, can detect the process drift in the first attack). In relation to the known attacks, the results demonstrate that our approach can in principle detect *direct* control modifications (our first attack closely resembles the corresponding semantic attacks in [10, 14, 26]). In the testbed we also show detecting indirect control attacks (the second attack manipulates measurements). However, as any operational ICS setting will pose more challenges than our small controlled environment, we analyse the feasibility of our approach with data from two real-world plants in §5.

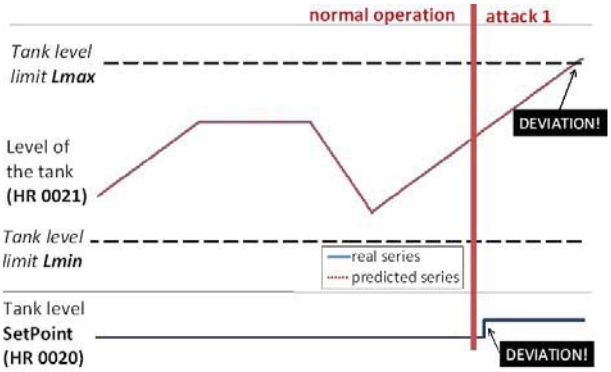

Figure 2: Illustration of the configuration change

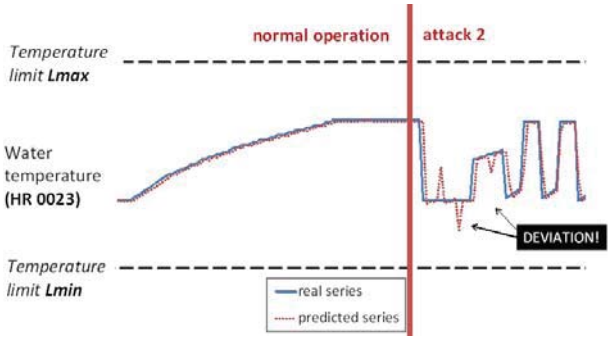

Figure 3: Illustration of measurement tampering

4. IMPLEMENTATION

We implemented a prototype of our approach using a combination of Bro [31] and custom C++ code. Bro performs the initial *data extraction* step. We developed a Modbus analyser for Bro that extracts the main protocol commands from network traffic and makes them available to scripts written in Bro's custom policy language. We leverage Bro's BinPAC [30] parser generator to automatically generate much of the Modbus-specific code from a corresponding grammar. The Bro project has integrated our Modbus analyser into Bro 2.2. We use a custom Bro script to record the parsed Modbus commands into an ASCII-based log file, which we then process with external C++ code implementing the characterisation, modelling, and detection phases. After a preliminary analysis testing different values of k for the data characterisation phase, we choose $k = 3$ for our tests with real environments as that value showed the least mismatches. For modelling continuous series, we build the autoregressive model and derive process control limits. We leverage an open source implementation of the autoregressive model.[1] To estimate autoregressive coefficients, we use Burg's method [19]. To estimate the order of the model, we use the common Akaike information criterion [34]. As control limits, we implement Shewart control limits following the description in [38]. For each continuous series we derive and estimate the behaviour with both the autoregressive and control limit models. Finally, for detecting deviations in continuous series via the autoregressive model, we use two variance hypothesis tests (commonly known as F-test). For both tests we set $p = 0.05\%$ as their significance level.

[1]Available at https://github.com/RhysU/ar.git

5. EVALUATION

Our work represents an initial step towards modelling the semantics of process variables from a network vantage point. To the best of our knowledge, we offer the first study analysing, at this level, week-long network traces from operational plants, where each PLC operates with thousands of process variables. As such, we are primarily interested in understanding properties of the setting that impact semantic approaches to security monitoring, and less in specific true/false positives rates. With that perspective, we evaluate our memory map modelling with two overarching objectives: *(i)* understand the degree to which our approach can successfully predict typical process behaviour; and *(ii)* gain insight into the underlying activity that improves, or weakens, its accuracy. In particular, we do not aim to evaluate detection rates, due to the inherent difficulty of achieving meaningful results in a realistic setting. As process attacks remain both novel and rare, we cannot expect our traces to contain malicious activity (and as far as we know, they do not). Yet it also remains unrealistic to inject crafted attacks into the traces, as we would remain limited to trivial, *direct control* cases like those already demonstrated in §3.5—which our detector would find for the same reasons as discussed there. It is hard to see how one could possibly inject *indirect control* attacks into existing, static traces, as these would normally trigger a chain of effects that only a full simulation of the plant could reproduce.

In the following we first introduce our two real-world environments in §5.1. Then in §5.2 we present the results of the characterisation and modelling phases. We do not analyse in depth the data extraction here as it constitutes primarily a pre-processing step.

5.1 Environments and Data Sets

Our data comes from two real-life water treatment plants that serve a total of about one million people in two urban areas; they are part of a larger system of over 30 sites controlled by one company. The two plants are of comparable size, and they perform semantically similar tasks (e.g., water pumping, purification, ozone treatment). Nevertheless their setups look quite distinct. They deploy different numbers of PLCs (3 vs. 7), and chose different strategies to divide processes among them. The PLC memory maps differ both between the two environments, and also among PLCs of the same site. While both plants use equipment from the same vendor, they deploy different software versions.[2] We have access to one 3-day and one 14-day long packet trace from the two plants, respectively, both containing the complete network traffic captured from mirroring ports of switches that connect the PLCs and the ICS servers. The traces include 64GB and 101GB of network traffic, respectively, with bandwidths varying between 9 and 360 packets/sec during the recorded periods. We find two ICS protocols in use: Modbus (between PLCs and ICS servers) and a vendor proprietary protocol (between ICSs server and HMI). In addition, we see six further non-ICS protocols in use by servers and workstations: VNC, SSL, FTP, HTTP, SMB, and DCOM. The non-ICS protocols constitute an negligible fraction of the overall network trace. Of the 20 and 28 hosts active in the two traces, 7 and 11 exchange Modbus messages. While we see the vendor proprietary protocol in use among hosts that are part of the supervisory infrastructure, we observe only Modbus for all communication involving PLCs. We see three types of read/write Modbus messages in the traces (function codes 3, 16 and 23).

To present our results, we select 5 PLCs taken from both plants, namely: all 3 PLCs from the first plant, and 2 PLCs from the second plant. The second plant operates with 7 PLCs in total, of differing complexity: the number of process variables goes as low as 135 for three of them and as high as 3500 in one; each of the remaining four PLCs operates with approximately 2200 variables. We select two PLCs representing the most simple and the most complex setup, respectively. The plant operators also provided us with *project files* that describe each PLC's memory map layout, exported by PLC programming environments in the form of CSV files holding information on addressing, data type, and process role for each process variable defined by a PLC. These files closely resemble the information shown by the example in Table 2, as well as in [32]. We do not use the project files for modelling but, instead, leverage them for explaining observed deviations at a later stage.

5.2 Data Characterisation and Modelling

First we run the characterisation (described in §3.4) on 3 days of network traffic from the first plant's PLCs. We find that it classifies 95.5% of all variables as constant series, 1.4% of variables as attribute, and 3.1% as continuous series. Further analysis with different batch choices shows that these results remain consistent over time for any intervals longer than one day. Next, to evaluate our modelling approach, we first generally measure the quality of our models. In the second, more interesting step, we then dig deeper into the results and focus on understanding the underlying reasons and situations in which our approach *(i)* indeed models process activity correctly; and *(ii)* fails to capture the plant's behaviour, flagging benign deviations as alarms. Our objective here is to gain insight into the capabilities and limitations of our approach applied in real life environments, as well more generally into potential and challenges of modelling process activity at the semantic level.

Generally, we consider our approach to generate an "alert" on a process variable when, at any time during testing a batch of data, an observation deviates from the prediction—i.e., when observing an unexpected value for constants and attributes, or a value outside the autoregressive/control limit models for a continuous time series. We perform 3-fold cross validation using the *rolling forecasting procedure* [20] on a set of 3-day batches of data extracted from the two plant's network traces. The first two batches of data come from the first plant, each representing a randomly chosen continuous range from the first and the second weeks, respectively. The third batch represents the complete trace from the second plant (recall that we only have 3 days of network traces available from that plant). At a technical level, a 3-day batch size gives us a reasonable volume of data suitable for processing repeatedly with our implementation. At an operational level, operators confirm to us that one day matches a typical PLC work cycle.

In Table 3 we summarize the results of the testing across different behaviour models.

[2]According to agreements with the two sites, we cannot name the equipment vendor.

Table 3: Testing model capabilities

	Deviating variables across different types of series (mean %/ st.dev)		
	Constant	*Attribute*	*Continuous*
PLC 1a	0.5/0.29	19.05/0.2	57.49/5.78
PLC 1b	0.31/0.04	19.80/1.4	44.64/5.41
PLC 1c	0.14/0.02	19.55/4.0	37.58/2.98
PLC 2a	0.64/0.0	26.92/0.0	63.63/0.0
PLC 2b	0.0/0.0	0.0/0.0	0.0/0.0

For each pair of category and PLC, we compute the percentage of alerts, showing mean and standard deviation over the batches. In the following, we discuss the three categories separately.

5.2.1 Constant Series

Our results show that by far the most variables that we classify as constant indeed stay stable over time. This corroborates the general perception that ICS configurations do not change often. Examining the small number of deviating variables (0.0% to 0.6%) in this category, we observe two main causes: *(i)* configuration changes, and *(ii)* misclassifications of the variable type. The former confirms that our approach indeed detects *direct* control modifications. While in this case the changes prove legitimate, they remain sufficiently rare that an IDS can generally report them for human inspection (or whitelisting). Examining the reported situations in more detail, we see that 60 variables all trigger at the same time due to an update to a "configuration matrix", a large data structure that defines an operation mode in terms of a set of values controlling multiple devices simultaneously (as it turns out, it is a single packet from the HMI that performs a "multiple register write"). We confirm that this is the only example of such a large configuration change in the analysed trace.

The second cause for unexpected deviation represents shortcomings of our data classification phase. For example, during one of the folds in PLC2a we find that it misclassifies 9 out of 15 measurement variables representing aggregated flow information as constant due to a lack of activity during the training period. Similarly, in the same fold we find that 7 out of 18 device statuses (which represent attribute data) get misclassified as constants. In both cases the values remain constant for more than 20 hours, yet then change during the testing period. Interestingly, we see several such situations that first trigger an alert for a configuration change (e.g., the status of a filter in PLC1b changes for 15mins after it has spent 21 hours in a previous state), followed by a burst of further ones reflecting the change being in effect now (i.e., variables representing activity linked to that filter start to deviate from constant behaviour: status, volume, throughput/hour, total throughput). In this case, the two main causes of errors are hence linked semantically. As others have observed, such *sequential variations* [26] represent cases that one could handle specifically, for example by merging into a single event.

Discussions with the plant operators confirm that daily *cleaning* activities on that PLC might cause such sudden changes for a short amount of time. It turns out that the next fold avoids this misclassification due to its longer training interval, confirming that when training spans the corresponding work cycle, the modelling of constants indeed works as expected.

5.2.2 Attribute Data Series

Our test shows a higher (0.0% to 26%) but stable number of deviating attribute series across all tested folds. By sampling a subset we find that their main cause of alerts concerns continuous variables which, due to slow process character, exhibit only a limited number of distinct values during a training interval, and are thus wrongly labelled as attribute data (e.g., variables describing time information in the form of date and hour). Apart from this special case, variables describing alarms and statuses do not report any alerts (i.e., there are no new types of alarms and events in the analysed trace).

With attribute data, sequences can carry important semantic information, as such variables often encode the current process state within a series of steps (e.g., alarm type X raised to operator, operator acknowledged, alarm cleared, state normal). Omitting specific steps can represent a threat, however the current analysis cannot capture such cases and applying a continuous model is counterproductive: while for attribute data ordering matters, timing does typically not (e.g., an operator may acknowledge an alarm at any time). In §5.3 we present an approach for further improving the analysis of attribute series.

5.2.3 Continuous Data Series

We now summarize results from the two models that consider continuous time series: control limits and autoregression (with deviations ranging between 0.0% and 63%). We observe that the autoregressive model generally alerts more frequently than control limits. In fact, the control limits contribute to only 28% of all deviations in continuous series.

Control Limits model.

Our results show that, apart from the overlap with the autoregressive model, control limits report an additional 3% of the series as deviating. Our inspection reveals that these variables represent increasing trends over the course of the whole trace. For such variables, approaches in statistical process control commonly accept that the series should be modelled according to their regression nature only, and not with control limits. This means that these variables should be whitelisted in this model. An alternative approach would be to obtain absolute process limits (e.g., from process engineers) and enforce those limits for series control.

Autoregressive model.

Our results show that the autoregressive model has no difficulty in modelling internal process stages and counter variables. Different from alarms and commands, which occur in relation to human interaction, these variables originate from automated process behaviour with highly correlated and regular sequences of values, which are straight-forward to capture. The results indeed show that automated process logic can be consistently monitored. Of all the deviations, we only find one related to a counter variable that delayed an increment for 2 seconds. While most likely such a deviation represents just a process irregularity, it could also indicate a replay attack as performed by Stuxnet.

The remaining deviations refer to high precision measurement variables. By inspecting them more closely, we can distinguish two groups. A first group of deviations refers to variables that autoregression fails to model well, independently of the training interval. This group accounts for

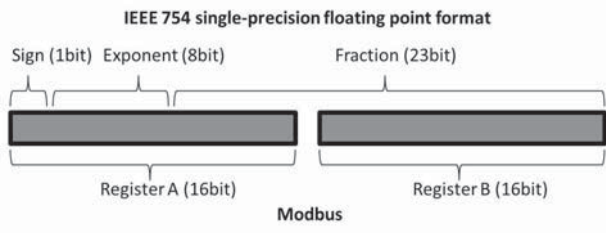

Figure 4: Representation of single precision floating point in Modbus

$\sim$ 80% of the deviations (consisting of the same variables reported consistently across all folds and batches). By sampling these variables, we find 70% exhibiting apparently random behaviour with high oscillations, while the remaining 30% remain nearly constant (or exhibit slow trends) with then sudden peaks. Referring to the project files, it turns out that they all correspond to *floating point* measurement values from the same set of field devices (e.g., measurements from devices concerning purification in PLC2a). Modbus represents floating point values with a set of registers. The vendor specification for our PLCs states that a single precision floating point is encoded according to the IEEE 754 standard [3] in *two* registers, which represent the actual value as a product of sign, exponent and mantissa. Figure 4 shows how these three components map onto a pair of registers. Taken independently, the two variables describing the behaviour of the two registers look quite different; while one variable can appear random, the second variable looks nearly constant. To address this problem, we would need to reconstruct the floating-point values as part of the data extraction phase. Unfortunately, it is technically challenging to identify the two halves of a floating point variable, as vendors use different approaches and even within the same vendor, programmers might follow different conventions due to the flat structure of the Modbus data model. Indeed, in the analysed PLC project files we observe the use of at least three different conventions: use consecutive pairs of registers with *(i)* the register with the even address as the upper register, *(ii)* the register with the odd address as the upper register, or *(iii)* for a set of variables, put all upper registers first and then all the lower registers. Working with a more structured protocol than Modbus (e.g., with IEC 60870-5-104) would help solve this problem.

The second major group of deviations concerns variables that behave differently over time. For example, one value remains nearly constant for 20 hours, then it fluctuates for 15 minutes and then it reverts back to a constant. This group of variables is the same group which was mischaracterised as constant series, as we describe in §5.2.1. The data series of these variables is not stationary in a wide sense, and thus it is generally not well suited for autoregressive modelling. The observed deviation further confirms that a unified approach can hardly address variables with different nature, origin and role. However, constituting a relatively small subset (up to 10 out of 2200 variables in one PLC), these could be treated as a special case.

5.3 Exploring Improvements

Summarizing the deviations from previous sections, we notice three distinct cases that impair our results: *(i)* characterisation issues, *(ii)* difficulties modelling attribute data, and *(iii)* dependencies amongst variables.

First, we observe that a large fraction of the reported deviations in Sections §5.2.1–5.2.3 come from the characterisation phase. A potential approach for introducing more semantics into the characterisation phase is the inclusion of knowledge contained in the project files. To explore the feasibility of this approach in practice, we translate the semantic information of the available project files into labels that define what we expect the characterisation phase to return. We assign the labels semi-automatically by constructing a table that maps keywords commonly found in the descriptions into the three categories. For example, we consider descriptions including "measurement", "counter", or "usage" to indicate variables holding continuous values; words like "command" or "alarm" suggest attribute data; "configuration" indicates a constant value. In total, we identify 24 keywords, which allow us to classify all PLC variables defined in the project files. Our analysis shows that we can cover 35% of all observed variables that way, with the main limiting factor being implicit definitions of multiple variables (e.g., PLC programmers use tailored data structures to define a range of variables by only defining the starting variable in project files). Comparing the labels with our heuristics, we see an excellent match for constant variables. However, only about half of the continuous variables match, and even less in the attribute category (around 30%). Closer inspection reveals as the main problem ambiguities in the project file that mislead the keyword-based heuristic. Generally, the descriptions are not standardised but depend on the PLC programmer, and hence keywords sometimes overlap. For example, one PLC has several fields that include the description "ControlForAlarm". Yet, we consider the keyword "control" to indicate a constant variable, and "alarm" to suggest an attribute series. While this example could be addressed easily, similar ambiguities would remain. This difficulty shows that in practice, even if project files are available, it is not straight-forward to extract meaningful semantic information from them.

Second, in §5.2.2 we point out that the current approach cannot capture the semantics or sequence of some attribute series (e.g., process alarms). However, during focus group sessions with plant engineers, we learned that alarms and commands are typically encoded in bitmaps, and we indeed find this reflected in the network traffic. For example, for a variable that the PLC project file refers to as *"various status notifications from PLC3 to server"* , we observe a series of what, at first, appears like an arbitrary set of values: 40960, 36864, 34816. However, when aligned in binary format, the values map to:

```
1010 0000 0000 0000
1001 0000 0000 0000
1000 1000 0000 0000
```

This representation reveals patterns of bits that are constant (e.g., the first bit indicates that PLC1c is active). If we integrated this structure into the characterisation step, we could refine the attribute modelling significantly. In other words, some variables require a different granularity than just their numerical value for capturing their semantics.

Third,we find distinct groups of variables that alarm at the same time. In all analysed cases, the deviations correspond to a limited set of devices with the behaviour that is *(i)* related (e.g., a peak of 9 deviating variables in PLC 2a semantically describe different aspects of only one field device, a pump) or *(ii)* identical (e.g., we see 10 ozone filters

whose flow is described by the same autoregressive model, and whose deviation occurred at the same time and thus resulted in a peak of deviations in one PLC). In either case, our preliminary analysis shows that the variables could be clustered (e.g., by performing per "point-in-time" vertical analysis) and presented as aggregated events.

6. RELATED WORK

A set of prior work focuses on understanding ICS communication patterns, showing that communication flows indeed reflect the regular and (semi-)automated character of process control systems [6].

Other efforts focus on analysing security threats in ICS. For example, some authors analyse protocol vulnerabilities [1, 4, 7], explore the lack of compliance to protocol specifications in different PLCs [9, 35], and analyse communication patterns for anomaly detection [22, 36]. However, the effects that one can find at the flow-level remain limited; detecting semantic process changes requires inspection of the application layer. Consequently, some authors propose to parse network protocols for extracting information that can highlight changes to the process environment. For example, authors perform partial protocol parsing to enumerate functionality that Modbus clients use, aiming to detect unexpected deviations in requests sent to PLCs [12], interpret events at a higher level [17], fingerprint current device configuration remotely [29, 33], and propose approaches for verifying PLC software [27, 28]. Düssel et al. [13] propose using application syntax (not *semantics*) for network-based anomaly detection, but do not further examine ICS-specific protocols.

In terms of classic IDS signatures, DigitalBond provides Snort preprocessors that add support for matching on Modbus/DNP3/EtherNetIP protocol fields [2].

McLaughlin and McDaniel in [24, 26] leverage PLC code and process descriptions to reconstruct the mapping of process variables in PLCs and thus craft efficient process attacks. In terms of analysis, this work is complementary to ours—the approach leverages PLC code while we use passive network monitoring to reconstruct process semantics. Conceptually, detecting attacks generated via SABOT [26] are within the scope of our approach. We show that key process variables (i.e., configuration settings) can be consistently monitored via our approach.

In [25] the authors present stateful policy enforcement which validates commands sent to physical devices from a PLC. While this approach can work even when the PLC gets compromised, the requirement of expert involvement is inevitable (while our approach requires little or no involvement).

Different authors analyse false data attacks against state estimation (thus targeting *indirect control* attacks) [5, 11, 23]. Similarly, yet from a network perspective, others evaluate accuracy of detectors on crafted attacks [16] and preconfigured undesirable states [15] within controlled testbeds on a set of selected process measurements. These works focus on understanding the accuracy of the modelling approach and leverage detailed specifications on the behaviour of the measurement variables in a controlled environment (e.g., including the specification of the process and state estimation).

By contrast, we perform an exploratory study to understand the behaviour of *all* process variables (i.e., both field and internal PLC variables) in a real world plant, providing insights into challenges for building automated process-aware, yet specification-agnostic, security detectors.

7. CONCLUSION

Our work ventures much deeper into ICS network traffic than prior efforts have offered. We monitor the communication of PLCs in two real-world water plants, extracting and modelling operational semantics at the level of individual process variables, to then flag unexpected deviations for manual inspection. Our motivation comes from recent *direct* process control attacks that aim to change static configuration parameters. Our approach detects these by watching for changes to variables that would normally stay constant. While that also reports legitimate changes, these generally either remain sufficiently rare—and, hence, an alert can act as a useful double-check—or amendable to whitelisting by staff if more frequent.

Going beyond static variables, we show that our approach can also detect *indirect* process control attacks by modelling the expected behaviour of non-constant variables. In practice, this proves more difficult, though, as real-world environments exhibit plenty of irregularities, semantic mismatches, and corner-cases that need care to get right. We offer a first step into that direction by focusing for now less on measuring detection rates, and more on gaining insight into conceptual capabilities and limitations. Consequently, we identify a range of reasons for our simple classifiers to fail in capturing dynamic process variables reliably, including mismatches between training and activity cycles, failure to correctly dissect a variable's data representation, and manual human intervention with the process. However, we also find that once we understand the root cause for a failure, we can often lay out a way forward to overcome that particular shortcoming (e.g., by exploiting the specific structure of a bitfield).

From a different perspective, when we derive models for thousands of real-world PLC variables, we find that indeed about 95% represent generally static configuration settings that our approach covers well. Including non-static variables, our approach manages to consistently build models of over 98% of the process variables. However, out results also suggest that in practice, due to their individual nature, one cannot expect a small set of generic classifiers to reliably capture them all equally well. We hence deem it more realistic to tailor a detector's focus to a smaller set of *high-value* variables that are *(i)* critical for the process and *(ii)* provide a complementary picture to any existing safety monitoring. As particularly interesting candidates we identify clusters of related variables that represent semantic dependencies within a process—existing safety systems do typically not consider them as a group.

Generally, we also plan to leverage further context to model individual process variables more tightly, including from more structured protocols (e.g., DNP3 and IEC 60870-5-104 define specific types for setpoints, counters, etc.) and further plant configuration files (e.g., Substation Configuration Language (SCL) files as used in IEC 61850 networks, ICS vendor project files). While generally the specific choice of variables that prove most valuable to monitor will remain site-specific, we see an economy of scale at a different level: one can formalise, and then replicate, the *procedure* of identifying and modelling suitable targets in hitherto unknown network environments.

Acknowledgments

This work has been partially supported by the European Commission through projects FP7-SEC-285477-CRISALIS and FP7-SEC-607093-PREEMPTIVE, funded by the 7th Framework Program; as well as by the U.S. National Science Foundation under grant CNS-1314973. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors or originators, and do not necessarily reflect the views of the sponsors.

8. REFERENCES

[1] Project basecamp. http://www.digitalbond.com/tools/basecamp/.

[2] Quickdraw SCADA IDS. http://www.digitalbond.com/tools/.

[3] IEEE Standard for Floating-Point Arithmetic. Technical report, Microprocessor Standards Committee of the IEEE Computer Society, 3 Park Avenue, New York, NY 10016-5997, USA, Aug. 2008.

[4] Common cybersecurity vulnerabilities in industrial control systems. U.S. Department of Homeland Security, 2011.

[5] S. Alajlouni and V. Rao. Anomaly Detection in Liquid Pipelines Using Modeling, Co-Simulation and Dynamical Estimation. In *Critical Infrastructure Protection VII*, volume 417 of *IFIP*, pages 111–124. Springer Berlin Heidelberg, 2013.

[6] R. R. Barbosa, R. Sadre, and A. Pras. Difficulties in modeling SCADA traffic: a comparative analysis. In *Proc. of the 13th international conference on Passive and Active Measurement*, PAM'12, pages 126–135, Berlin, Heidelberg, 2012. Springer-Verlag.

[7] C. Bellettini and J. L. Rrushi. Vulnerability analysis of SCADA protocol binaries through detection of memory access taintedness. In *Proc. 8th IEEE SMC Information Assurance Workshop*, pages 341–348. IEEE Press, 2007.

[8] G. E. P. Box and G. Jenkins. *Time Series Analysis, Forecasting and Control*. Holden-Day, Incorporated, 1990.

[9] E. Byres, D. Hoffman, and N. Kube. On shaky ground - a study of security vulnerabilities in control protocols. In *NPIC HMIT*, 2006.

[10] A. Carcano, I. N. Fovino, M. Masera, and A. Trombetta. Critical information infrastructure security. chapter Scada Malware, a Proof of Concept, pages 211–222. Springer-Verlag, Berlin, Heidelberg, 2009.

[11] 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 *Proc. of the 6th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '11, pages 355–366, New York, NY, USA, 2011. ACM.

[12] S. Cheung, B. Dutertre, M. Fong, U. Lindqvist, K. Skinner, and A. Valdes. Using model-based intrusion detection for SCADA networks. In *Proceedings of the SCADA Security Scientific Symposium*. Digital Bond, 2007.

[13] P. Düssel, C. Gehl, P. Laskov, J.-U. Busser, C. Stormann, and J. Kastner. Cyber-critical infrastructure protection using real-time payload-based anomaly detection. In *Proc. of the 4th international conference on Critical information infrastructures security*, CRITIS'09, pages 85–97, Berlin, Heidelberg, 2010. Springer-Verlag.

[14] N. Falliere, L. O. Murchu, and E. Chien. W32.Stuxnet Dossier-symantec security response. [online], 2011.

[15] I. N. Fovino, A. Carcano, M. Masera, A. Trombetta, and T. Delacheze-Murel. Modbus/DNP3 state-based intrusion detection system. In *Proc. of the 2010 24th IEEE International Conference on Advanced Information Networking and Applications*, AINA '10, pages 729–736, Washington, DC, USA, 2010. IEEE Computer Society.

[16] W. Gao, T. Morris, B. Reaves, and D. Richey. On SCADA Control System Command and Response Injection and Intrusion Detection. In *eCrime Researchers Summit (eCrime)*, 2010.

[17] J. Gonzalez and M. Papa. Passive scanning in Modbus networks. In *Critical Infrastructure Protection*, volume 253 of *IFIP International Federation for Information Processing*, pages 175–187, 2007.

[18] D. Hadžiosmanović, L. Simionato, D. Bolzoni, Z. Zambon, and S. Etalle. N-Gram Against the Machine: On the Feasibility of the N-Gram Network Analysis for Binary Protocols. In *Proc. Research in Attacks, Intrusions, and Defenses*, volume 7462 of *LNCS*, pages 354–373. Springer Berlin Heidelberg, 2012.

[19] M. Hoon. Parameter Estimation of Nearly non-Stationary Autoregressive Processes. Technical report, Delft University of Technology, 1995.

[20] R. J. Hyndman and G. Athanasopoulos. Forecasting: principles and practice. An online textbook.

[21] P. International. Profinet application layer service definition, 2004. V. 1.95.

[22] O. Linda, T. Vollmer, and M. Manic. Neural network based intrusion detection system for critical infrastructures. In *Neural Networks, IJCNN International Joint Conference on*, pages 1827 –1834, june 2009.

[23] Y. Liu, P. Ning, and M. K. Reiter. False data injection attacks against state estimation in electric power grids. In *Proc. of the 16th ACM conference on Computer and communications security*, pages 21–32. ACM, 2009.

[24] S. McLaughlin. On dynamic malware payloads aimed at programmable logic controllers. In *Proc. of the 6th USENIX conference on Hot topics in security*, HotSec'11, pages 10–10, Berkeley, CA, USA, 2011. USENIX Association.

[25] S. McLaughlin. Cps: Stateful policy enforcement for control system device usage. In *Proc. of the 29th Annual Computer Security Applications Conference*, ACSAC '13, pages 109–118. ACM, 2013.

[26] S. McLaughlin and P. McDaniel. SABOT: specification-based payload generation for programmable logic controllers. In *ACM Conference on Computer and Communications Security*, pages 439–449. ACM, 2012.

[27] S. McLaughlin, Z. S., D. Pohly, and P. McDaniel. A Trusted Safety Verifier for Process Controller Code. In *In Proc. Network and Distributed System Security Symposium*. The Internet Society, 2014.

[28] S. Mohan, S. Bak, E. Betti, H. Yun, L. Sha, and M. Caccamo. S3A: Secure System Simplex Architecture for Enhanced Security of Cyber-Physical Systems. *CoRR*, abs/1202.5722, 2012.

[29] P. Oman and M. Phillips. Intrusion detection and event monitoring in SCADA networks. In *Critical Infrastructure Protection*, volume 253 of *IFIP*, pages 161–173. Springer US, 2007.

[30] R. Pang, V. Paxson, R. Sommer, and L. Peterson. binpac: A yacc for Writing Application Protocol Parsers. In *ACM Internet Measurement Conference*, 2006.

[31] V. Paxson. Bro: a system for detecting network intruders in real-time. *Computer Networks*, 31(23-24):2435–2463, 1999.

[32] Schneider Electric. *Modbus Protocol and Register Map for ION Devices*, 70022-0124-00 edition.

[33] R. Shayto, B. Porter, R. Chandia, M. Papa, and S. Shenoi. Assessing the integrity of field devices in Modbus networks. In *Critical Infrastructure Protection II*, volume 290 of *IFIP*, pages 115–128. Springer Boston, 2009.

[34] N. Sugiura. Further analysts of the data by Akaike' s information criterion and the finite corrections. *Communications in Statistics - Theory and Methods*, 7(1):13–26, 1978.

[35] A. Treytl, T. Sauter, and C. Schwaiger. Security measures for industrial fieldbus systems - state of the art and solutions for IP-based approaches. In *Proc. IEEE International Workshop on Factory Communication Systems*, pages 201 – 209, 2004.

[36] A. Valdes and S. Cheung. Communication pattern anomaly detection in process control systems. In *Proc. of International Conference on Technologies for Homeland Security*, Waltham, MA, May 11–12, 2009. IEEE.

[37] C. Vulnerabilities and Exposures. CVE-2010-4709 - heap-based buffer overflow in automated solutions Modbus/TCP Master OPC Server. http://www.cvedetails.com/cve/CVE-2010-4709/, 2011.

[38] G. B. Wetherill and D. W. Brown. *Statistical process control: theory and practice*. Chapman and Hall, London, New York, 1991.