Modeling Modbus TCP for Intrusion Detection

Mustafa Faisal[1], Alvaro A. Cardenas[1], and Avishai Wool[2]

Abstract—**DFAs (Deterministic Finite Automata) and DTMCs (Discrete Time Markov Chain) have been proposed for modeling Modbus/TCP for intrusion detection in SCADA (Supervisory Control and Data Acquisition) systems. While these models can be used to learn the behavior of the system, they require the designer to know the appropriate amount of training data for building the model, to retrain models when configuration changes, and to generate understandable alert messages. In this paper, we propose to complement these learned models with the specification approaches. To build a robust model, we need to consider configuration-level specifications in addition to protocol specification. As Modbus/TCP is a simple protocol with handful function code(s) or commands for each communication channel, designing a specification-based approach is suitable for monitoring this communication. We do a comparison of DFA and DTMC approaches in two datasets and illustrate how to use our inferred specification to complement these models.**

I. Introduction

To facilitate the detection of anomalies and potential attacks in industrial control networks, various approaches have been proposed in the literature for modeling the communication channels between Human Machine Interfaces (HMIs) and Programmable Logic Controllers (PLCs). Models of HMI and PLC communication channels include Deterministic Finite Automata (DFA) [1] and its variants ([2], [3]), and Discrete-Time Markov Chains (DTMCs) [4]. DFA and DMTC models are automatically learned form *normal* network traffic in the industrial network and are expected to detect deviations from this normal behavior, and thus, serving as an Intrusion Detection System (IDS).

Learning-based models face several challenges: For example, DFA models do well as long as the communication patterns are periodic, however, non-periodic patterns and multiple patterns will introduce complexity. Moreover, this model suffers from inability to flag delay attacks. On the other hand, DTMC models are expensive with more states and transitions. Finally, both models can be exploited with contaminated (malicious) learning data, either from the beginning or when the system updates its configuration and needs to learn a new model.

In this paper, we analyze these issues and propose a specification-based intrusion detection as a complimentary approach. Modbus TCP is a relatively simple protocol and a HMI-PLC communication channel generally does not include a large number of function codes from this protocol, therefore, building protocol and configuration-level specifications for each channel is feasible.

[1] Mustafa Faisal and Alvaro A. Cardenas are with the Department of Computer Science, The University of Texas at Dallas (UT Dallas). The work at UT Dallas was supported by NSF grant CNS-1547502.

[2] Avishai Wool is with the School of Electrical Engineering, Tel Aviv University (TAU). Supported in part by a grant from the Interdisciplinary Cyber Research Center at TAU.

II. Related Work

Specification-based intrusion detection has been previously considered by Cheung *et al* [5]. In this work we explore in more depth the specification process, configuration-level specification, and how specification compares to learning-based approaches.

Morris et al. [6] developed some of the first rules for detecting intrusions specifically in Modbus and Modbus TCP networks. That same year Goldenberg *et al* [1] proposed a way to automatically learn a DFA for Modbus TCP; in particular, they determine a periodic pattern of symbols which can be used as a reference for detecting anomalies. Subsequent work extends this approach for the Siemens S7 protocol [2]. Kleinman *et al* [3] extended the approach to statecharts that allow for multi-threaded HMIs generating multiplexed channel pattern. Recently Barbosa et al. [7] proposed an tool named "PeriodAnalyser" with three modules: Multiplexer - separates communication traffic into various flows, Tokenizer - transforms each packet into a protocol-independent format (token), and Learner - processes each token to find and characterize periodic activities. Finally, Caselli *et al.* [4] proposes another way to deal with the multi-threaded and non-periodic nature of some industrial networks by developing sequence-aware IDS based on probabilistic DTMC models.

While developing an IDS for industrial control networks might be relatively easier than developing an IDS for general information technology networks, these previous works illustrate that developing models to automatically capture the operations of an industrial network still face several challenges.

III. Background

A. Modbus TCP

Modbus TCP is one of the variants of basic Modbus, a popular communication protocol in SCADA systems. Modbus TCP clients and servers listen and receive data through port 502. This protocol [8] is composed by an Application Data Unit (ADU) which consists of two parts: Modbus Application Protocol (MBAP) header and Protocol Data Unit (PDU), as illustrated in figure 1.

MBAP Header: The MBAP header has four fields: *Transaction Identifier (TID)* is used for pairing transactions because multiple messages can be transmitted through the same TCP connection by a client without waiting for a prior response; *Protocol Identifier (PID)* is always 0 for Modbus; *Length* provides the length of the following fields (in bytes) which includes unit identifier (UID), function code, and data fields; and *Unit Identifier* is used to identify a remote PLC located on a non-TCP network (for serial bridging). By default, UID is set to 00 or FF which is ignored by the server and echoed back in the response.

PUD: Three types of PDUs are defined in Modbus [9]: Modbus request, response, and exception response. PDUs

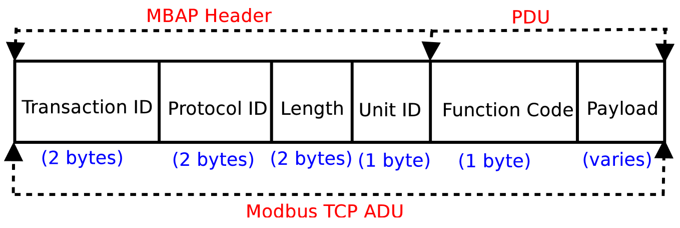

Fig. 1. Modbus TCP data packet

have two fields, i) *Function Code* (for Modbus request and response PDU) or exception function code (for Modbus exception response—where the most significant bit is set as 1, and ii) *Payload*—from 1 to 252 bytes—which can be a variable reference number (RN), count, values, data offsets, sub-function codes, etc.

Function code: In Modbus, valid function codes are from 1 to 127 and divided into three categories: public (well documented), user-defined (vendor-specific functions), and reserved (used by legacy products). Public function codes [9] are $\{1 - 8, 11 - 12, 15 - 17, 20 - 24, 43\}$.

Modbus data model: There are four primary tables which are organized in series: discrete input, coils, input registers, and holding registers. Depending on the function code in the Modbus request, access is directed to a specific primary table. And based on the device, these tables can be organized in four separate blocks or a single block. In a Modbus PDU, a data item can be addressed from 0 to 65,535. In addition, based on Modbus request and response, fields may vary. For example, Modbus read request data has two fields: RN and bit/word count where first one specifies the memory address to start reading and second one specifies the quantity of memory object unit to be read. In the corresponding Modbus response the payload that does not contain RN, rather it has i) byte count—amount of complete response data and ii) data—values of the memory objects that were read.

B. Specification-based Intrusion Detection for Modbus: Benefits and Challenges

In a specification-based IDS, a model of the specifications or requirements are constructed, and any deviation from this expected normal behavior will raise alarms. An specification-based IDS is essentially a "whitelist" of allowed behaviors or connections in the network. Unlike learning-based anomaly detection methods, a specification is usually obtained from a design document or by a manual description of desired behavior. Describing a priori all the allowed behaviors of a system might be a challenge and it might produce general an permissive systems, or tightly controlled systems that generate many false alarms. In some cases building specifications is expensive and unmanageable [10]. In addition, verification of the model (constructed from this approach) is also a challenging task.

On the other hand, because Modbus is a simple protocol and in most cases, a few function code(s) are used in each HMI-PLC channel. Building specifications for each channel is relatively easy. Moreover, knowledge of the specification can be reused in various channels. In addition, because we are defining allowed behavior a priori, even if the specification or configuration of the system changes, we can generally extract this new specification from the design documents to deploy new specification-based IDS on day one, while learning-based models would need to wait until we capture data from the new configuration to train the new classifiers.

To build a tailored specification for a specific industrial deployment, we cannot rely on protocol-level specification because it would be easy for attackers to find new attacks while satisfying the Modbus TCP protocol specification. To build a specification capturing the details of individual deployments we need to focus on the configuration of industrial systems. For example, while the protocol specification will remain the same for a specific function code, based on specific requirements, the starting address and quantity of data may change from channel to channel; so if two channels A and B carry data with only function code 3 (Read Holding Registers) their model can be totally different because of different starting addresses and output quantities. If we do not consider these details, an adversary can send request commands with any undesired valid function code, starting address, and quantity; or she can also jam or delay the response and model.

IV. Our Approach

We consider rules into two categories: rules derived from the protocol, and rules derived from the configuration specification. The rules are as follows:

1) The function code should be valid for a particular channel. In a channel, only a subset of function codes are used.
2) Starting address or RN and quantity of outputs should be within valid ranges.
3) For a specific channel, RN as well as quantity of outputs should be matched with predefined values.
4) The wait time for a response should be within a predefined time duration.

Rule 1 is derived from both the protocol and configuration specification. Rule 2 is based on protocol specification and rules 3 and 4 are related to configuration.

In rule 1, a function code should be valid according to the protocol specification. And for a specific channel, a defined valid set of function code(s) are allowed. This set can be modified due to change in requirements. The reference number, according to rule 2, should be within a valid range for the used function code. And the same rule also enforces valid output quantities so that total addresses do not exceed the valid range of the address space; the valid address range and output quantities for each public function code can be found in the Modbus TCP protocol specification. Rules based on configuration-specific requirements like rule 3, ensure that the starting address and quantity of outputs matched their configured values of function codes for a specific channel. Finally, rule 4 ensures that the server should not wait for a long time for the reply of a valid request. IDS rules are generated from channel specific configuration files.

V. Datasets

We use two datasets: dataset #1 is a one-day trace from a real-world operational large-scale water facility in the U.S. and dataset #2 (used by [1]) is collected from the facility manager of Tel Aviv University which monitors campus power grid using Modbus TCP.

A brief summary of these datasets is provided in table I.

Data Set#	Description	Size(GB)	# Packet	AER	# Channels
1	Water treatment plant	10	802,392.02	802	94
2	University campus power grid	6.2	48,835,082	673	6

TABLE I
OVERVIEW OF DATASETS. HERE AVERAGE EVENT RATE (AER) IS CALCULATED AS NUMBER OF EVENTS OR PACKETS PER SECOND

In our datasets, different kinds of function codes are used, as listed in table II. Note that (as far as we are aware) there is no attack during the network trace capture.

Data Set#	type of function code	Function Codes
1	read	1,2,3,4
	write	15,16
	read/write	23
2	read	1,2,3
	write	5

TABLE II
USED FUNCTION CODES IN OUR DATASETS

VI. EXPERIMENTS AND RESULTS

In this paper, we compare our specification with the single DFA of Goldenberg *et al.* [1] and the DTMC of Caselli *et al.* [4]. Our specification was implemented as rules for each communication channel in the *Bro* network monitoring tool. For the other two approaches, data is extracted by *Pyshark*, a Python tool for *tshark* used in *Wireshark*. Extracted data is stored in the file system and *MySQL*. Symbols are created with *JAVA* and *MySQL* queries. Models are implemented in *JAVA*.

Results from DFA: In a DFA, an input symbol is defined as a 4-tuple (with 33-bits) $< Query/Response(1\ bit)$, $function\ code(8\ bits)$, $reference\ number(16\ bits)$, $bit/word\ count(8\ bits)\ >$. Note that response messages do not have reference number and so this 16-bit section remains zero in response messages. We use a Moore DFA to model where transition functions take a base state with an input symbol and return a destination state with a corresponding action. In [1], four types of transition functions are defined: **normal** (match with next expected state in pattern), **re-transmission** (base and destination states are the same), **miss** (mismatch with the expected state), and **unknown** (appearance of an unknown symbol) transition.

A learning algorithm determines whether there is a smallest-size DFA to model the channel's communication trace. For a given length ($Pattern\_length$) of a candidate DFA, performance is defined as:

$$\frac{\#normal}{\#normal + \#re-transmission + \#miss + \#unknown}$$

for the validation dataset (where #x represents the number of x events in the dataset).

This learning algorithm takes $Learning\_window\_length$ (a fixed number of packets in the network trace for selecting a candidate DFA with size $Pattern\_length$), $Validation\_window\_length$ (a fixed number of packets in the network trace for validating a candidate DFA), and a threshold as parameters. The authors select $threshold = \frac{n}{n+1}$, where n is the length of candidate DFA. The algorithm iteratively checks if there is a smallest $Pattern\_length$ of DFA for which a performance value exceeds the threshold for the provided $Validation\_window\_length$ data. If

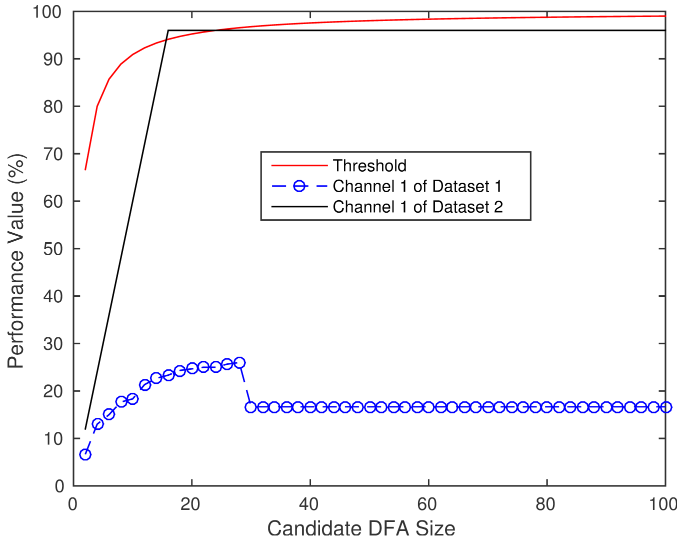

Fig. 2. Performance (%) vs. candidate DFA size for channel #1 of dataset #1 and dataset #2 respectively

the algorithm does not find any acceptable DFA within a $Learning\_window\_length$, it returns "failed". The algorithm returns a DFA when the performance value exceeds the threshold for the candidate DFA. In our experiments, we set $Learning\_window\_length$=100 and $Validation\_window\_length$=300. Like [1], we also set $Pattern\_length$=2 and increments by 2 until we reach $Learning\_window\_length$.

For dataset #1, we have 94 channels with over 1000 packets. Only 28% (27 out 94) produce DFAs above the threshold value (i.e., 68% of the channels cannot be modeled with this DFA learning algorithm). We observe that the data size of the successful channels is relatively small (the largest one is 11 MB) while the maximum-sized channel among all is about 165 MB and the average data size per channel is about 25 MB.

For dataset #2 all channels resulted in successful DFA models (we got the same results reported by [1]).

Data Set#	Channel #	Function Code	# Pair
1	1	3	1,229,088
2	1	1	484,499
		2	900
		3	3,392,428
		5	1

TABLE III
STATISTICS OF FUNCTION CODES USED IN TWO SELECTED CHANNELS

To illustrate some of the differences we select two channels from these datasets one has a clear pattern (channel 1 of dataset 1) and the other has a non-periodic pattern (channel 1 of dataset 2). Statistics of used function codes for selected channels are presented in table III. The performance values for the candidate DFAs are illustrated in figure 2. We can see that channel #1 of dataset #1 produces DFAs whose performance value (blue circled line) is always below the selection threshold for all candidate DFA sizes. However, for channel #1 of dataset #2 (black line), we obtain a successful DFA (with size 16) guaranteeing that this DFA is a good model of the interactions in that channel. Understanding why the DFA

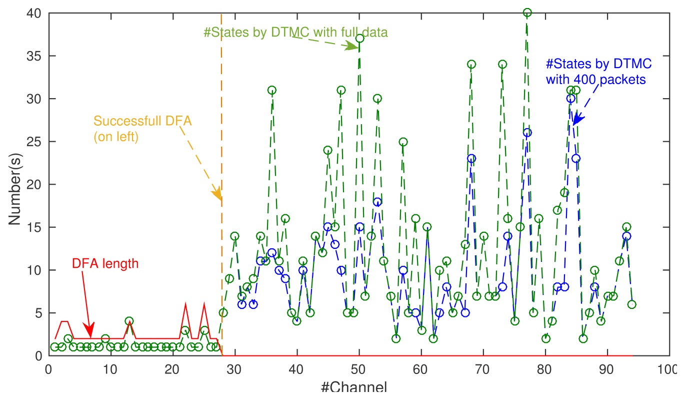

Fig. 3. DFA length (in red) as well as #states generated by 400 packets (blue) and full data (green) respectively for 94 channels of dataset # 1.

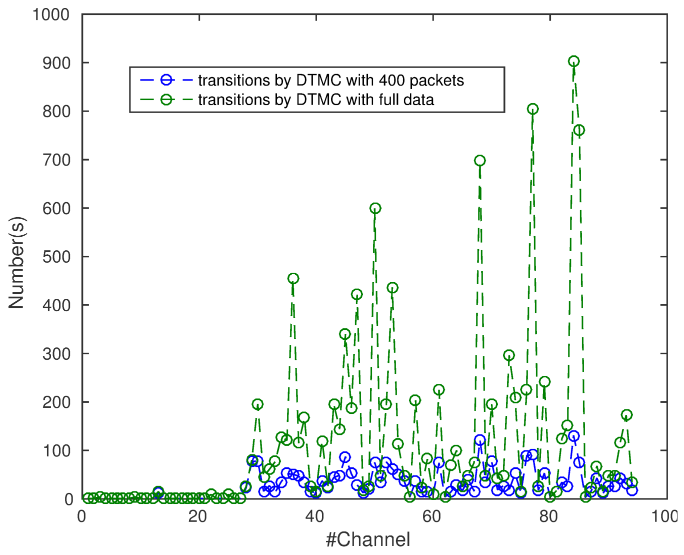

Fig. 4. # of transitions generated by 400 packets (blue) and full data (green) respectively of 94 channels by dataset #1.

model is unsuccessful for 68% of the channels in dataset #1 is left for future work.

Results from DTMC: In DTMC, a state, S, is defined by a 5-tuple: $< Data,\ Type,\ number\ of\ events,\ First\ Time\ Seen,\ Last\ Time\ Seen >$ and every transition, T (from a source state to a destination state), is defined by a 6-tuple: $< probability,\ number\,of\,jumps,\ first\,jump,\ last\,jump,\ average\,time\,elapsed,\ standard\,deviation\,on\,time\,elapsed >$. In the learning and evaluation stage, states and transitions are constructed with training data. A detection mechanism is used to find *unknown state, unknown transition*, and *unknown probability* in test data with respect to the model built in training phase.

The number of (states, transitions) for our two selected channels from datasets #1 and #2 are (15, 225) and (57, 259) respectively (using all our data). However, if we use only 400 packets (as used for learning and validation in the DFA-based algorithm), we obtain (15, 89) and (9, 18) respectively (note that if we use all data for DFA training our results don't change significantly, only one more channel becomes unable to be modeled as a DFA due to repair-induced anomalies in the dataset). Interestingly, some states are only "request" or "response" types for channel #1 of dataset #2. This is mainly because of timing in the network trace capture—at the start of the trace, some responses were collected without request packets and at the end of the trace, some requests were collected without responses. For example we got only one unpaird TID for channel 1 of dataset #1 but 49,153 unpaired TIDs for channel 1 of dataset #2.

Comparison: Figure 3 illustrates the size of successfully-generated DFAs and the number of states generated by DTMCs for the 94 channels of dataset # 1. Recall that we were only able to model 27 channels from dataset #1 with a DFA; the sizes of the DFAs learned in these 27 channels are shown in red on the left side of the dashed yellow line. We can see that the length of the successful DFAs is quite small (between 1 and 6).

On the right side of the yellow line are channels that could not be modeled with the DFA algorithm. Notice that the number of states needed by DTMCs to model these channels grows significantly because these channels have patterns that are more complex to learn. To learn these DTMC models we used all our training data, and 400 packets to illustrate that a small number of channels can be modeled quite effectively with these initial packets, while other channels exhibit different behaviors after these 400 packets and thus need to be trained for longer periods of time.

In figure 4 we show the number of transitions among the states in DTMC using 400 packets and full data of 94 channels in dataset #1.

Implementing the Specification: We implemented our specification rules in *Bro*. For Modbus, *Bro* provides function-code specific events. For example, for function code 3, there are two events: *modbus_read_holding_registers_request* and *modbus_read_holding_registers_response*. Rule 1 (regarding valid function code) is implemented in event *modbus_message* where we check whether a channel carries data with a non-configured function code. Rules 2, 3 and 4 are implemented in function-code-specific request and response events. We set a maximum response time of 5 seconds; any longer delay will raise an alert. Using this setting we got 15 alerts from delayed responses in channel 1 of dataset #1, and 0 alarms for channel 1 of dataset #2.

In principle a specification should be expressed a priori from the designers of the system or should be extracted from configuration files in the system, unfortunately for our dataset we do not have either of these resources. We were allowed to collect packet traces from industrial networks but we were not allowed to access configuration files. As such to create the specification rules we looked at the data traffic and inferred from the dataset. These rules can be complementary to the models from DFA and DTMC and provide a more holistic anomaly-detection approach to intrusion detection in control systems.

VII. Discussion

A. Result analysis

The DFA-based approach is a good way to capture the dynamics of simple channels, but it faces challenges when modeling more complicated communication channels. The

DTMC approach can model all channels in our datasets, but the increasing number of states and transitions for complex patterns can be an issue. For channels where we obtained a successful DFA, the number of states and transitions generated by DTMC are smaller than channels for which we obtained the "failed" result. The states number and size of DFA are similar for channels with successful DFAs. For these channels, the same number of states and transitions are generated with 400 packets or the full dataset. However, this is not true for channels with failed DFAs.

B. Comparison with various approaches

Learning and evaluation Cost: Each symbol in a DFA requires 33 bits. The number of transitions like normal, miss, re-transmission, etc. are calculated to find the performance value. A learning algorithm iteratively validates each DFA candidate until the performance value exceeds the threshold or the pattern length exceeds the learning window. Thus, model size can be bounded within a learning window. The evaluation phase is simple and less expensive than the training phase as all we need to compare is the network trace with a DFA.

We need to store a state with 5 attributes and a transition with 6 attributes (with different size and data types) for the DTMC approach. The number of states depends on the appearance of various commands in the training data and can be bounded with a number of unique commands in data. The transaction number in this approach varies and depends on frequency of moves from one state to another. Here the training and testing phases are identical since in both cases we need to find the states and transitions among the states. The deviation from training to testing phase in terms of state, transition, and probability should be determined. Thus, this approach requires considerable amount of computation and storage in comparison with DFAs. And here the model construction may be affected by the amount of training data.

For our specification-based approach, we need to find a way to read the configuration for Modbus TCP for each channel and then build rules based on protocol specification and configuration. We need expert knowledge for the protocol and its implementation in the system. Configuration files may be written in different formats (as Modbus is an open-source protocol and deployment approaches of this protocol can vary) but the required attributes can be found in the protocol specification. Given these configuration files, we do not need to collect training or testing data.

Model update: The configuration may change as time passes. Both DFA and DTMC-based approaches need new data for re-training and evaluating the new models for each channel. For the specification-based approach, whenever there is a change in the configuration file-that will be reflected in the channel rule file.

Meaningful alert message for operator: Learning-based approaches like DFA and DTMC do not generate very specific alerts or logs for operators to understand. For example, a DFA IDS may raise alarms for unknown transitions, but this may be difficult for operators to understand or to know which field or attribute associates to the alert that would help her to take a countermeasure. In the same way, DTMC unknown states, transitions, or probabilities may not be meaningful for operators. On the other hand, the specification can generate alerts with more information derived from the rules. For example, if there is an appearance of a non-configured function code in the channel, the IDS will raise an alarm and log the reason in a way which would help the operator understand and solve the problem.

C. Implementation issues of specification-based approach

As mentioned earlier, we need to build a rule file to monitor each channel where a set of configurations are stored in a configuration file. A script can be written to gather configured values, and an event handler will monitor whether there is a change in the configuration file for Modbus TCP. Whenever there is a change in a configuration file, this should lead to a change in the rule file for the IDS.

VIII. Conclusion and future work

In this paper, we analyze and discuss challenges of approaches to model network channels between the HMI and PLCs for Modbus TCP. As a complementary approach to capture Modbus TCP characteristics that can be missed by DFA and DTMCs, we propose a specification-based IDS and we show that the protocol specification is not enough to model this channel, rather we need to consider configuration-level specification.

In future work we will look in more detail at the reasons DFAs fail to model some channels and if DTMCs are good enough to cover these failures. We will also look in more detail at what adversaries can do to evade DFAs, DTMCs, and specification rules. Finally we would like to consider other industrial protocols in future work.

References

[1] N. Goldenberg and A. Wool, "Accurate modeling of modbus/tcp for intrusion detection in scada systems," *International Journal of Critical Infrastructure Protection*, vol. 6, pp. 63–75, 2013.

[2] A. Kleinman and A. Wool, "Accurate modeling of the siemens s7 scada protocol for intrusion detection and digital forensics," *The Journal of Digital Forensics, Security and Law: JDFSL*, vol. 9, p. 37, 2014.

[3] A. Kleinmann and A. Wool, "A statechart-based anomaly detection model for multi-threaded scada systems," in *International Conference on Critical Information Infrastructures Security*, 2015, pp. 132–144.

[4] M. Caselli, E. Zambon, and F. Kargl, "Sequence-aware intrusion detection in industrial control systems," in *Proceedings of the 1st ACM Workshop on Cyber-Physical System Security*, 2015, pp. 13–24.

[5] 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*, vol. 46, 2007, pp. 1–12.

[6] T. H. Morris, B. A. Jones, R. B. Vaughn, and Y. S. Dandass, "Deterministic intrusion detection rules for modbus protocols," in *System Sciences (HICSS), 2013 46th Hawaii International Conference on*, 2013, pp. 1773–1781.

[7] R. R. R. Barbosa, R. Sadre, and A. Pras, "Exploiting traffic periodicity in industrial control networks," *International journal of critical infrastructure protection*, vol. 13, pp. 52–62, 2016.

[8] M. Organization, "Modbus Messaging on TCP/IP Implementation Guide V1.0b," http://www.modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf, 2012, [Online; accessed 05-July-2016].

[9] ——, "Modbus Application Protocol Specification V1.1b3," http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf, 2012, [Online; accessed 05-July-2016].

[10] R. Berthier and W. H. Sanders, "Specification-based intrusion detection for advanced metering infrastructures," in *Dependable Computing (PRDC), 2011 IEEE 17th Pacific Rim International Symposium on*, 2011, pp. 184–193.