A Statechart-Based Anomaly Detection Model for Multi-Threaded SCADA Systems

Amit Kleinmann(✉) and Avishai Wool

Tel-Aviv University, 69978 Tel-aviv, Israel
amitkl@post.tau.ac.il, yash@eng.tau.ac.il

Abstract. SCADA traffic between the Human Machine Interface (HMI) and the Programmable Logic Controller (PLC) is known to be highly periodic. However, it is sometimes multiplexed, due to asynchronous scheduling. Modeling the network traffic patterns of multiplexed SCADA streams using Deterministic Finite Automata (DFA) for anomaly detection typically produces a very large DFA, and a high false-alarm rate. In this paper we introduce a new modeling approach that addresses this gap. Our *Statechart DFA* modeling includes multiple DFAs, one per cyclic pattern, together with a DFA-selector that de-multiplexes the incoming traffic into sub-channels and sends them to their respective DFAs. We evaluated our solution on traces from a production SCADA system using the Siemens S7-0x72 protocol. We also stress-tested our solution on a collection of synthetically-generated traces. In all but the most extreme scenarios the *Statechart* model drastically reduced both the false-alarm rate and the learned model size in comparison with the naive single-DFA model.

1 Introduction

1.1 Background

SCADA systems are used for monitoring and controlling numerous Industrial Control Systems (ICS). In particular, SCADA systems are used in critical infrastructure assets such as chemical plants, electric power generation, transmission and distribution systems, water distribution networks, and waste water treatment facilities. SCADA systems have a strategic significance due to the potentially serious consequences of a fault or malfunction.

SCADA systems typically incorporate sensors and actuators that are controlled by Programmable Logic Controllers (PLCs), and which are themselves managed by a Human Machine Interface (HMI). PLCs are computer-based devices that were originally designed to perform the logic functions executed by electrical hardware (relays, switches, and mechanical timer/counters). PLCs have evolved into controllers with the capability of controlling the complex processes used for discrete control in discrete manufacturing.

This work was supported in part by a grant from the Israeli Ministry of Science and Technology.

E. Rome et al. (Eds.): CRITIS 2015, LNCS 9578, pp. 132–144, 2016.
DOI: 10.1007/978-3-319-33331-1_11

SCADA systems were originally designed for serial communications, and were built on the premise that all the operating entities would be legitimate, properly installed, perform the intended logic and follow the protocol. Thus, many SCADA systems have almost no measures for defending against deliberate attacks. Specifically, SCADA network components do not verify the identity and permissions of other components with which they interact (i.e., no authentication and authorization mechanisms); they do not verify message content and legitimacy (i.e., no data integrity checks); and all the data sent over the network is in plaintext (i.e., no encryption to preserve confidentiality). Therefore, deploying an Intrusion Detection Systems (IDS) in a SCADA network is an important defensive measure.

1.2 Related Work

Byres et al. [5] describe different attack trees on SCADA systems based on the Modbus/TCP protocol. They found that compromising the slave (PLC) or the master (HMI) has the most severe potential impact on the SCADA system. For instance, an attacker that gains access to the SCADA system could identify as the HMI and change data values in the PLC. Alternately, an attacker can perform a Man In The Middle attack between a PLC and HMI and "feed" the HMI with misleading data, allegedly coming from the exploited PLC.

Carcano et al. describe a system with a pipe in which flows high pressure steam [12]. The pressure is regulated by two valves. An attacker capable of sending packets to the PLCs can force one valve to complete closure, and force the other to open. Each of these SCADA commands is perfectly legal when considered individually, however when sent in a certain order they bring the system to a critical state. Marsh [18] presents an attack scenario where a system-wide water hammer effect is caused simply by opening or closing major control valves too rapidly. This can result in a large number of simultaneous main breaks. The Stuxnet malware [11,17] implemented a similar attack by changing centrifuge operating parameters in a pattern that damaged the equipment - while sending normal status messages to the HMI to hide the fact that an attack is under way.

Fundamentally all these attacks work by injecting messages into the communication stream—possibly legitimate messages—on an attacker-selected pattern and schedule. Hence a good anomaly detection system needs to model not only the messages in isolation but also their sequence and timing.

A survey of techniques related to learning and detection of anomalies in critical control systems can be found in [2].

While most of the current commercial network intrusion detection systems (NIDS) are signature-based, i.e., they recognize an attack when it matches a previously defined signature, Anomaly-based Network Intrusion Detection Systems (IDS) "are based on the belief that an intruder's behavior will be noticeably different from that of a legitimate user" [19].

Different kinds of Anomaly Intrusion Detection models have been suggested for SCADA systems. Yang et al. [26] used an Auto Associative Kernel Regression (AAKR) model coupled with the Statistical Probability Ratio Test (SPRT) and

applied them on a SCADA system looking for matching patterns. The model used numerous indicators representing network traffic and hardware-operating statistics to predict the 'normal' behavior. Several recent studies [3,7] suggest anomaly-based detection for SCADA systems which are based on Markov chains. However, Ye et al. [27] showed that although the detection accuracy of this technique is high, the number of False Positive values is also high, as it is sensitive to noise. Hadiosmanovic et al. [14] used the logs generated by the control application running on the HMI to detect anomalous patterns of user actions on process control application.

Nai Fovino et al. [12] have presented a state-based intrusion detection system for SCADA systems. Their approach uses detailed knowledge of the industrial process' control to generate a system virtual image. The virtual image represents the PLCs of a monitored system, with all their memory registers, coils, inputs and outputs. The virtual image is updated using a periodic active synchronization procedure and via a feed generated by the intrusion detection system (i.e., known intrusion signatures).

Model-based anomaly detection for SCADA systems, and specifically for Modbus traffic, was introduced by Cheung et al. [8]. They designed a multi-algorithm intrusion detection appliance for Modbus/TCP with pattern anomaly recognition, Bayesian analysis of TCP headers and stateful protocol monitoring, complemented with customized Snort rules [21]. In subsequent work, Valdes and Cheung [23] incorporated adaptive statistical learning methods into the system to detect for communication patterns among hosts and traffic patterns in individual flows. Later Briesemeister et al. [4] integrated these intrusion detection technologies into the EMERALD event correlation framework [20].

Sommer and Paxson [22] discuss the surprising imbalance between the extensive amount of research on machine learning-based anomaly detection pursued in the academic intrusion detection community, versus the lack of operational deployments of such systems. One of the reasons for that, by the authors, is that the machine learning anomaly detection systems are lacking the ability to bypass the "semantic gap": The system "understands" that an abnormal activity has occurred, but it cannot produce a message that will elaborate, helping the operator differentiate between an abnormal activity and an attack.

Erez and Wool [10] developed an anomaly detection system that detects irregular changes in SCADA control registers' values. The system is based on an automatic classifier that identifies several classes of PLC registers (Sensor registers, Counter registers and Constant registers). Parameterized behavior models were built for each class. In its learning phase, the system instantiates the model for each register. During the enforcement phase the system detects deviations from the model.

Goldenberg and Wool [13] developed a model-based approach (the GW model) for Network Intrusion Detection based on the normal traffic pattern in Modbus SCADA Networks.

Subsequently, Kleinmann and Wool [16] demonstrated that a similar methodology is successful also in SCADA systems running the Siemens S7 protocol.

Caselli et al. [6] proposed a methodology to model sequences of SCADA protocol messages as Discrete Time Markov Chains (DTMCs). They built a state machine whose states model possible messages, and whose transitions model a "followed-by" relation. Based on data from three different Dutch utilities the authors found that only 35 %–75 % of the possible transitions in the DTMC were observed. This strengthens the observations of [13,16] of a substantial sequentiality in the SCADA communications. However, unlike [13,16] they did not observe clear cyclic message patterns. The authors hypothesized that the difficulties in finding clear sequences is due to the presence of several threads in the HMI's operating system that multiplex requests on the same TCP stream. Each independently scheduled thread is responsible for certain intervals of registers.

1.3 Contributions

DFA-based models have been shown to be extremely effective in modeling the network traffic patterns of SCADA systems [13,16], thus allowing the creation of anomaly-detection systems with low false-alarm rates. However, the existing DFA-based models can be improved in some scenarios.

In this paper we address two such scenarios: the first scenario is the one identified in [6]: the HMI is multi-threaded, each thread independently scans a separate set of control registers, and each thread has its own scan frequency. The second scenario occurs when the SCADA protocol allows the HMI to "subscribe" to a certain register range, after which the PLC asynchronously sends a stream of notifications with the values of the subscribed registers. The commonality between the scenarios is that the network traffic is not the result of a single cyclic pattern: it is the result of several multiplexed cyclic patterns. The multiplexing is due to the asynchronous scheduling of the threads inside the HMI, or to the asynchronous scheduling of PLC-driven notifications. Attempting to model a multiplexed stream by a single DFA typically produces a very large DFA (it's cycle length can be the least-common-multiple of the individual cycle lengths), and also a high false-alarm rate because of the variations in the scheduling of the independent threads.

Our solution to both scenarios is the same: instead of modeling the traffic of an HMI-PLC channel by a single DFA, we model it as a *Statechart* of multiple DFAs, one per cyclic pattern, with a DFA-selector that de-multiplexes the incoming stream of symbols (messages) into sub-channels and sends them to their respective DFAs. Our design supports simple cases, in which each sub-channel has a unique set of symbols—and also the complex cases in which the patterns overlap and some symbols belong to multiple sub-channels.

We evaluated our solution on traces from a production SCADA system using the latest variant of the proprietary Siemens S7 protocol, so called S7-0x72. Unlike the standard S7-0x32 protocol, which is fairly well understood, little is published about the new variant. Based on recent advances in the development of an open-source Wireshark dissector for this variant, we were able to model S7-0x72 in the *Statechart* framework, including its subscribe/notify capability. A naive single-DFA model caused a false-alarm rate of 13–14 % on our traces,

while the *Statechart* model reduced the false-alarm rate by two orders of magnitude, down to at most 0.11 %. A separate contribution is our description of the S7-0x72 protocol, with its complex message formats and advanced semantics.

We also stress-tested our solution on a collection of synthetically-generated traces, with intentionally difficult scenarios multiplexing up to 4 periodic patterns and with up to 56 % symbol overlap between patterns. In all but the most extreme scenarios the *Statechart* model drastically reduced both the false-alarm rate and the model size in comparison with the naive single-DFA model.

2 The DFA-based Model for Modbus

The GW model [13] was developed and tested on Modbus traffic. Modbus is a simple request-response protocol widely used in SCADA networks. A Modbus HMI sends a request to a Modbus PLC. The request includes a function code specifying the service, and the address range of data items. Modbus functions include reading values from coils (bit-size entities) or registers (16-bit entities), writing values to coils and registers, and performing diagnostics. After the PLC processes the request, it sends a response back to the HMI.

In the GW model, the key assumption is that traffic is *periodic*, therefore, each HMI-PLC channel is modeled by a Mealy Deterministic Finite Automaton (DFA). The DFA for Modbus has the following characteristics: (a) A symbol is defined as a concatenation of the message type, function code, and address range, totaling 33-bits; (b) A state is defined for each message in the periodic traffic pattern.

The GW model suggests a network anomaly detection system that comprises two stages: A learning stage, and an enforcement stage. In the learning stage a fixed number of messages is captured, the pattern length is revealed, and a DFA is built for each HMI-PLC channel. The learning assumes that the sniffed traffic is benign. In the enforcement stage, traffic is monitored for each channel (according to its DFA), and proper events are triggered.

Based on traffic captured from a production Modbus system, Goldenberg and Wool discovered that over 97 % of Modbus traffic is well modeled by a single DFA per HMI-PLC channel. However they also discovered a phenomenon that challenges the DFA-based approach: In addition to a frequent scan cycle that occurs multiple time per second, they found a second periodic pattern with a 15-minute cycle. Attempting to model both cycles by a single DFA produces a very large, unwieldy model: Its normal pattern consists of hundreds of repetitions of the fast scan cycle followed by one repetition of the slow cycle. Such a pattern is also inaccurate since the slow cycle does not always interrupt the fast cycle at the same point, and while the slow pattern is active, symbols from both patterns are interleaved.

3 A Statechart-Based Solution

Our first observation is that, as hypothesized by Caselli et al. [6] modern HMIs employ thread-based architecture (e.g., this is how the Afcon's Pulse HMI [1] is

built): While each thread is responsible for certain tasks (e.g., controlling access to a range of registers on a PLC), the threads run concurrently with different scheduling frequencies, and share the same network connections. Hence, to accurately model the traffic produced by such an HMI (with the PLC's responses), we should use a formalism that is more descriptive than a basic DFA. Our choice is to base our model on the concept of a Statechart [15]: the periodic traffic pattern driven by each thread in the HMI is modeled by its own DFA within the *Statechart*. Each DFA is built using the learning stage of the GW model. The *Statechart* also contains a DFA-selector to switch between DFAs.

3.1 The Statechart Enforcement Phase

During the enforcement stage, each DFA in the *Statechart* maintains its own state, from which it transitions based on the observed symbols (messages).

The DFA-selector's role is to send the input symbol s to the appropriate DFA. To do so it relies on a symbol-to-DFA mapping ϕ: $\phi(s)$ denotes the set of DFAs that have symbol s in their pattern. If each pattern has a unique set of symbols then ϕ is 1-1. However, in the general case, a symbol may appear in multiple patterns and ϕ is one-to-many. Upon receiving a symbol s the DFA-selector uses the following algorithm:

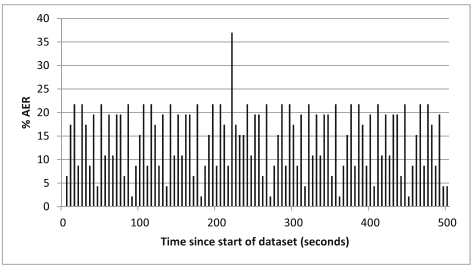
- If $\phi(s) = \varnothing$ the DFA-selector reports an "Unknown" symbol.
- If $\phi(s) = \{D\}$, i.e., the symbol is a unique symbol of a single DFA D, then s is sent to D, which handles it using its own transition function.
- Else, if $|\phi(s)| > 1$, the selected DFA is the member of $\phi(s)$ for which the absolute difference between the current time and the *predicted arrival time of* s is minimal.

In order to implement this policy:

- During the DFA learning stage of the GW model, for each state r in the DFA's pattern we calculate the average time difference to its immediate successor in the cyclic pattern (along the "Normal" transition). We denote this Time to Next State by $TNS(r)$.
- During the enforcement phase, each DFA D retains the time-stamp $T_{last}(D)$ of the last symbol that was processed by it (in addition to the identifier of the current state).

The predicted arrival time $T_{pred}(s, D)$ of a symbol s for a DFA $D \in \phi(s)$ which is currently at state q, is calculated as follows:

1. Identify the tentative state q' that DFA D transitions to from state q upon symbol s. Note that q' is not necessarily the immediate successor of q in the pattern—the transition from q to q' may be a "Miss" or a "Retransmission".
2. Let $P(q, q')$ denote the path of DFA states starting at q and ending at q' along the "Normal" transitions (not including q'). Then $T_{pred}(s, D) = T_{last}(D) + \sum_{r \in P(q,q')} TNS(r)$: The predicted arrival time is the sum of inter-symbol delays along the "Normal" path between q and the tentative transition-to state q' added to the time-stamp of the last symbol processed by DFA D.



(a) Applying the naive model on dataset #1.

Dataset #	1		2	
Duration	560 Sec.		2632 Sec.	
TCP Packets	15875		67585	
S7 Packets	4600		23553	
AER	9.19		9.16	
Dataset #	**1**		**2**	
DFA type	Naiv	Schrt	Naiv	Schrt
Model size	62	3	12	3
False alrm %	14.54	0.11	12.98	0

(b) Results of applying both models

Fig. 1. Detected abnormal symbols after applying the models on the S7 datasets.

3.2 The Statechart Learning Phase

The goal of the learning phase is to construct the *Statechart* for a specific HMI-PLC channel, given a captured stream symbols from the channel. For this we need to create the symbol-to-DFA mapping ϕ, for the use of the DFA selector, and we need to create the individual DFAs themselves. A key component in this learning phase is the Goldenberg and Wool learning algorithm, that accepts a periodic stream of symbols and creates a single DFA that best models that stream. Thus our *Statechart* learning phase is done as follows:

1. Split the channel's input stream into multiple sub-channels.
2. For each sub-channel use the GW learning algorithm to create a DFA.
3. Create the DFA-selector's mapping ϕ from the sub-channel DFAs.

The sub-channel splitting (step 1) can be implemented in different ways, depending on the available semantic knowledge. The easy case is when we know how many sub-channels can exist, each sub-channel has a unique set of symbols, and there is a filter criterion to recognize them. In this case the splitting algorithm works as a simple demultiplexer: for every input symbol it activates the filter criterion and sends the symbol to the (single) sub-channel based on the filter outcome. The difficult case is when we don't know in advance how many sub-channels exist, and the sub-channels potentially have overlapping symbols.

In the S7-0x72 traces we observed the easy case: the channel consisted of 2 sub-channels, one for request and response messages, and the other for notification messages. Since the message types are in the packet meta-data it is easy to split the input stream. Similarly, Goldenberg and Wool [13] reported that in their Modbus traces the slow and fast cycles had distinct symbols.

However, it seems that in the Modbus data set analyzed by Caselli et al. [6] the number of sub-channels is not clear in advance, and sub-channel symbols may be overlapping. Since this data set was not available to us we chose to stress-test the capabilities of our *Statechart* approach in this scenario using synthetic data (see Sect. 5.2).

4 The S7-0x72 Protocol

The S7 PLC Platform. The Siemens SIMATIC S7 product line is estimated to have over 30 % of the worldwide PLC market [9]. It includes both standard PLC models (S7-200, S7-300 and S7-400), and new generation PLCs (S7-1200 and S7-1500). Siemens has its own HMI software for its SIMATIC products called STEP7 and uses its own S7 communication protocol, over TCP port 102.

Two different protocol flavours are implemented by SIMATIC S7 products: The standard SIMATIC S7 PLCs implement a legacy S7 flavor, identified by the value 0x32, while the new generation PLCs implement a very different S7 flavor identified by 0x72. Among other changes, the newer S7-0x72 protocol also supports security features.

The standard S7-0x32 protocol is quite well understood, and a standard Wireshark dissector is available for it. The newer S7-0x72 protocol is not yet fully described in open literature. There is, however, a Wireshark dissector for it which is still in beta status [24].

A unique feature of the S7-0x72 protocol is its optional subscription model (in addition to the traditional request-response pattern). The HMI can send a special "subscribe" message, referring to certain control variables, to a PLC. Subsequently the PLC sends back a periodic stream of "notification" messages with the values of the subscribed variables. The challenge that this subscription model poses to a DFA-based anomaly detection system is that the notification messages are sent asynchronously, and are not part of the HMI-driven request-response pattern.

Experimenting with the S7-0x72 Data. Due to the proprietary nature and potential sensitivity of SCADA operations, real SCADA network data is rarely released to researchers. An important aspect of this work is that we were able to collect and analyze traces from a production S7 network running the S7-0x72 protocol from a control network of a solar power plant. In these traces we observed a single channel between the HMI and a Siemens S7-1500 PLC. We observed both the request-response and the unique subscribe/notification communication patterns. An overview of the S7 datasets can be found in Fig. 1b. During our recordings the infrastructure was running normally without any intervention of operators.

The message format and protocol semantics described here are based on the reverse engineering work of Wiens [24]. Somewhat surprisingly the S7-0x72 message formats are very different from those of the older S7-0x32 protocol, even though the overall protocol semantics are quite similar. An S7 0x72 packet is composed of the following parts:

- Header: 'magic ID' byte with a value of 0x72, a PDU type (one byte) and the length of the data part.
- Data part: includes meta data fields describing the data, data values, and an optional integrity part that is supported only by the newest S7-1500 PLCs (it contains two bytes representing an ID, one byte for the digest length and

a 32 byte message digest, which is apparently a cryptographic hash or MAC, details are yet unknown).
– Trailer: utilized to enable fragmentation.

Unlike the packet structure of the S7-0x32 protocol, nearly every field inside the S7-0x72 data part may be composed of recursively defined data structures. Further, elementary components such as numeric values are encoded using the variable-length quantity (VLQ) encoding [25], a universal code that uses an arbitrary number of binary octets. The precise S7-0x72 packet structure depends on the type of the command and the information it is instructed to carry. The beta Wireshark dissector [24] is able to parse the structure of over 30 different S7-0x72 commands.

To use the GW model we need to hash the meta-data fields of a SCADA packet into a symbol while ignoring the actual data values. In order to model the S7-0x72 packets we relied on the deep understanding embedded in the Wireshark dissector [24] to identify the structural meta-data components in the packets (command codes and arguments, register types and reference ids, etc.). In total we extracted 11–17 meta-data fields, comprising of 17–26 bytes, out of typical S7-0x72 packets, which were hashed into 64-bit symbols.

Figure 1a shows the false alarm rate over time of the naive DFA model applied to S7 dataset #1. Figure 1b summarizes the results on the two S7 traces, comparing the Naive and Statechart models. We can see that the naive DFA model has high false-alarm rates: 14.54 % and 12.98 %. The *Statechart* model successfully reduced the false-alarm rate by two orders of magnitude, down to at most 0.11 %. The table shows that the model sizes dropped from the incorrect sizes of 62 and 12 by the naive DFA model down to the correct size of 3 (a request-response pattern of 2 symbols and a notification pattern of 1).

Table 1. Overview of the sets of sequences used to generate the synthetic datasets

ID	Length	Uniq.	Period
1	6	6	300
	4	4	950
2	6	6	300
	4	4	950
3	6	4	300
	4	1	400
4	6	4	300
	4	2	950
5	10	9	300
	4	2	600
	4	3	200
6	10	7	300
	10	7	950
	10	7	2000
7	10	8	300
	8	7	350
	10	9	400
8	10	8	300
	8	7	850
	10	9	1300
9	10	7	300
	8	4	350
	10	8	400
10	6	3	300
	4	2	350
	6	2	400
11	10	8	250
	4	2	650
	6	4	1100
	8	7	420
12	6	4	250
	4	4	350
	10	9	550
	8	7	420
13	10	9	300
	4	2	600
	4	2	200
	6	3	350

5 Stress Testing with Synthetic Data

5.1 Generation of Synthetic Data

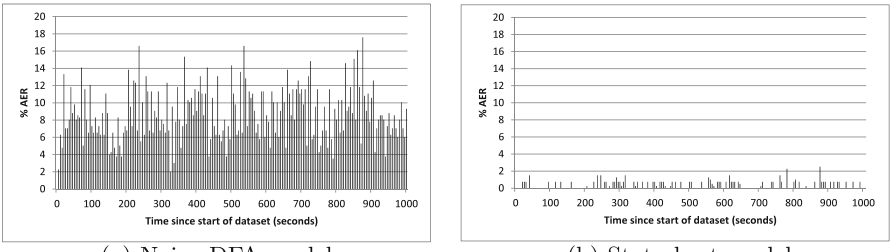
In order to test our model in different scenarios, we implemented a multi-threaded generator, where each of the threads simulates an HMI thread transmitting a cyclic pattern of SCADA commands. Each simulated thread has a pattern P of symbols, and a frequency f. Every f msec the thread wakes up and emits the pattern P as a burst, at a 1-msec-per-symbol rate, and returns to sleep. The thread's true timing has a jitter caused by the OS scheduling decisions. Further, when multiple threads are active concurrently then their emitted symbols are arbitrarily serialized.

We generated 13 scenarios, varying the number of patterns, the number of unique symbols per pattern, and their frequency. Table 1 shows the parameters of the scenarios that were used in our simulations. For the purpose of our evaluation and analysis we defined the following metrics:

- The *Symbol Uniqueness* of a channel $= \sum_{i=1}^{n} U_i / \sum_{i=1}^{n} L_i$, where L_i is the length of the cyclic pattern of sub-channel i and U_i is the number of symbols unique to that sub-channel.
- A channel's *Time Overlap* is the percentage of 1-msec time slots at which multiple packets where scheduled to be sent over the communication link during the time of the trace.
- The *model size* of a DFA is its number of states, and the model size of a statechart is the sum of the model sizes of its DFAs.

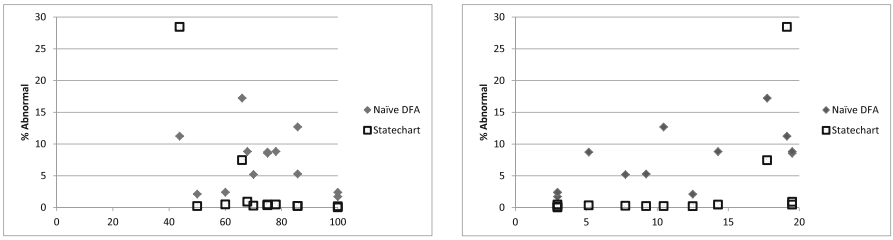
5.2 Experiments with the Synthetic Data

We started our evaluation by running the DFA described by Goldenberg and Wool, which we henceforth call the "naive-DFA". We ran the model's learning stage on the synthetic datasets with a maximum pattern length of 100 symbols



(a) Naive DFA model (b) Statechart model

Fig. 2. The false-alarm rate of the two models on synthetic dataset #11. Each time frame on the X axis represents 5 s. The Y axis shows the false alarm frequency as a percentage of the Average Event Rate (AER) for each time period.



(a) Symbol Uniqueness (%) (b) Time Overlap (%)

Fig. 3. The false alarm rates as a function of the Symbol Uniqueness and Time Overlap over the synthetic datasets.

and a validation window of 400 ($100 \cdot 4$) symbols. Then we ran the enforcement stage on the full datasets using the learned patterns.

When we applied the naive DFA model on the synthetic datasets it learned model sizes that are on average 3.5 times longer than the statechart model sizes for the same traces. Moreover, the *Statechart* model produced a much lower false-alarm rate on the same datasets. E.g, Fig. 2 illustrates the results of applying the two models on dataset #11.

Figure 3 shows that the *Statechart* managed to model the benign traffic successfully with very low false-alarm rate: up to 0.9% in nearly all our intentionally complex scenarios. The two exception cases are of datasets #10 (the worst result) and #13 (2nd worst result) that have very low symbol uniqueness (44 % and 67% respectively, compared to an average of 77% for the successful cases) and a high time overlap (19.13% and 17.74% respectively—approximately twice the average of the successful cases of 9.76%). In other words, only when around half of the symbols are not unique to a single pattern, *and* there is significant time overlap between patterns, does the *Statechart* model's performance deteriorate. In the more realistic scenarios, when symbol uniqueness is high or when the time overlap is low, the model performs extremely well.

6 Conclusions

In this paper we developed and applied the *Statechart DFA* model, which is designed specifically for anomaly detection in SCADA networks. This model has three promising characteristics. First, it exhibits very low false positive rates despite its high sensitivity. Second, it is extremely efficient: it has a compact representation, it keeps minimal state during the enforcement phase, and can easily work at line-speed for real-time anomaly detection. Third, its inherent modular architecture makes it scalable for protecting highly multiplexed SCADA streams. Our experiments demonstrate that the *Statechart DFA* anomaly detection model handles multiplexed SCADA traffic patterns very well.

References

1. Afcon Technologies: Pulse HMI Software (2015). Accessed 6 May 2015
2. Alcaraz, C., Cazorla, L., Fernández, G.: Context-awareness using anomaly-based detectors for smart grid domains. In: Proceedings of the 9th International Conference on Risks, and Security of Internet and Systems (CRISIS), Trento, Italy, September 2014
3. Atassi, A., Elhajj, I.H., Chehab, A., Kayssi, A.: The State of the Art in Intrusion Prevention and Detection, Auerbach Publications. In: Intrusion Detection for SCADA Systems, pp. 211–230. Auerbach Publications, January 2014
4. Briesemeister, L., Cheung, S., Lindqvist, U., Valdes, A.: Detection, correlation, and visualization of attacks against critical infrastructure systems. In: 8th International Conference on Privacy Security and Trust (PST), pp. 17–19 (2010)
5. Byres, E.J., Franz, M., Miller, D.: The use of attack trees in assessing vulnerabilities in SCADA systems. In: Proceedings of the International Infrastructure Survivability Workshop (2004)
6. Caselli, M., Zambon, E., Kargl, F.: Sequence-aware intrusion detection in industrial control systems. In: Proceedings of the 1st ACM Workshop on Cyber-Physical System Security, pp. 13–24. ACM, New York (2015)
7. Chen, C.-M., Hsiao, H.-W., Yang, P.-Y., Ya-Hui, O.: Defending malicious attacks in cyber physical systems. In: IEEE 1st International Conference on Cyber-Physical Systems, Networks, and Applications (CPSNA), pp. 13–18, August 2013
8. Cheung, S., Dutertre, B., Fong, M., Lindqvist, U., Skinner, K., Valdes, A.: Using model-based intrusion detection for SCADA networks. In: Proceedings of the SCADA Security Scientific Symposium, pp. 127–134 (2007)
9. Electrical Engineering Blog: The top most used PLC systems around the world. Electrical installation & energy efficiency, May 2013. http://engineering.electrical-equipment.org/electrical-distribution/the-top-most-used-plc-systems-around-the-world.html
10. Erez, N., Wool, A.: Control variable classification, modeling and anomaly detection in Modbus/TCP SCADA networks. In: 9th Annual IFIP Working Group 11.10 International Conference on Critical Infrastructure Protection, Washington, DC, USA, March 2015
11. Falliere, N., Murchu, L.O., Chien, E.: W32. stuxnet dossier. White Paper, Symantec Corporation, Security Response (2011)
12. Fovino, I.N., Carcano, A., De Lacheze Murel, T., Trombetta, A., Masera, M.: Modbus/DNP3 state-based intrusion detection system. In: 24th IEEE International Conference on Advanced Information Networking and Applications (AINA), pp. 729–736. IEEE (2010)
13. Goldenberg, N., Wool, A.: Accurate modeling of modbus/tcp for intrusion detection in SCADA systems. Int. J. Crit. Infrastruct. Prot. **6**(2), 63–75 (2013)
14. Hadziosmanovic, D., Bolzoni, D., Hartel, P.H., Etalle, S.: MELISSA: towards automated detection of undesirable user actions in critical infrastructures. In: Proceedings of the European Conference on Computer Network Defense, EC2ND 2011, Gothenburg, Sweden, pp. 41–48, USA, IEEE Computer Society, September 2011
15. Harel, D.: Statecharts: a visual formalism for complex systems. Sci. Comput. Program. **8**(3), 231–274 (1987)
16. Kleinmann, A., Wool, A.: Accurate modeling of the siemens S7 SCADA protocol for intrusion detection and digital forensic. JDFSL **9**(2), 37–50 (2014)

17. Langner, R.: Stuxnet: dissecting a cyberwarfare weapon. IEEE Secur. Priv. **9**(3), 49–51 (2011)
18. Marsh, R.T.: Critical foundations: protecting america's infrastructures - the report of the president's commission on critical infrastructure protection. Technical report, October 1997
19. Mukherjee, B., Heberlein, L.T., Levitt, K.N.: Network intrusion detection. IEEE Network **8**(3), 26–41 (1994)
20. Porras, P.A., Neumann, P.G.: EMERALD: event monitoring enabling responses to anomalous live disturbances. In: 1997 National Information Systems Security Conference, October 1997
21. Roesch, M.: Snort - lightweight intrusion detection for networks. In: Proceedings of the 13th USENIX Conference on System Administration, LISA 1999, pp. 229–238. USENIX Association, Berkeley (1999)
22. Sommer, R., Paxson, V.: Outside the closed world: on using machine learning for network intrusion detection. In: 2010 IEEE Symposium on Security and Privacy (SP), pp. 305–316, May 2010
23. Valdes, A., Cheung, S.: Communication pattern anomaly detection in process control systems. In: IEEE Conference on Technologies for Homeland Security (HST), pp. 22–29. IEEE (2009)
24. Wiens, T.: S7comm wireshark dissector plugin, January 2014. http://sourceforge.net/projects/s7commwireshark
25. Wikipedia: Variable-length quantity – Wikipedia, the free encyclopedia, (2015). Accessed 5 May 2015
26. Yang, D., Usynin, A., Hines, J.W.: Anomaly-based intrusion detection for SCADA systems. In: 5th Int International Topical Meeting on Nuclear Plant Instrumentation, Control and Human Machine Interface Technologies, pp. 12–16 (2006)
27. Ye, N., Zhang, Y., Borror, C.M.: Robustness of the markov-chain model for cyber-attack detection. IEEE Trans. Reliab. **53**(1), 116–123 (2004)