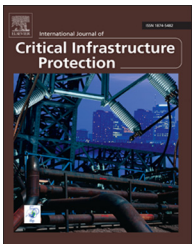
Available online at www.sciencedirect.com

SciVerse ScienceDirect

www.elsevier.com/locate/ijcip



Accurate modeling of Modbus/TCP for intrusion detection in SCADA systems

*Niv Goldenberg, Avishai Wool**

School of Electrical Engineering, Tel Aviv University, Ramat Aviv 69978, Israel

ARTICLE INFO

Article history:
Received 9 January 2013
Accepted 24 April 2013
Available online 4 May 2013

Keywords:
SCADA systems
Modbus/TCP
Network intrusion detection system

ABSTRACT

The Modbus/TCP protocol is commonly used in SCADA systems for communications between a human–machine interface (HMI) and programmable logic controllers (PLCs). This paper presents a model-based intrusion detection system designed specifically for Modbus/TCP networks. The approach is based on the key observation that Modbus traffic to and from a specific PLC is highly periodic; as a result, each HMI-PLC channel can be modeled using its own unique deterministic finite automaton (DFA). An algorithm is presented that can automatically construct the DFA associated with an HMI-PLC channel based on about 100 captured messages. The resulting DFA-based intrusion detection system looks deep into Modbus/TCP packets and produces a very detailed traffic model. This approach is very sensitive and is able to flag anomalies such as a message appearing out of its position in the normal sequence or a message referring to a single unexpected bit. The intrusion detection approach is tested on a production Modbus system. Despite its high sensitivity, the system has a very low false positive rate—perfect matches of the model to the traffic were observed for five of the seven PLCs tested without a single false alarm over 111 h of operation. Furthermore, the intrusion detection system successfully flagged real anomalies that were caused by technicians who were troubleshooting the HMI system. The system also helped identify a PLC that was configured incorrectly.

1. Introduction

Supervisory control and data acquisition (SCADA) systems are used for monitoring and controlling numerous industrial and infrastructure processes. 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 wastewater 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), which are themselves managed using a human–machine interface (HMI). 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).

*Corresponding author.
E-mail address: yash@acm.org (A. Wool).

1874-5482/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.ijcip.2013.05.001

Meanwhile, technological and economic trends have driven SCADA systems away from proprietary components and serial communications to off-the-shelf commodity components and IP-based communications protocols. The Modbus/TCP protocol is commonly used in SCADA networks for HMI-PLC communications. An attacker who injects malicious Modbus messages a SCADA network could cause significant damage. Therefore, deploying an intrusion detection system in a Modbus network is an important defensive measure.

This paper describes a model-based intrusion detection system designed specifically for Modbus/TCP networks. The detection approach is based on the fact that Modbus traffic to and from a specific PLC is highly periodic, with the same messages being sent repeatedly according to a fixed pattern. As a result, it is possible to model each HMI-PLC channel using its own unique deterministic finite automaton (DFA).

An algorithm is presented for automatically constructing a DFA associated with a HMI-PLC channel based on about 100 captured messages. The resulting DFA-based intrusion detection system looks deep into Modbus/TCP packets and produces a traffic model that captures detailed packet characteristics—not just function codes, but also the specific registers and coils referred to by messages. Based on the packet characterization, the model captures the precise periodic traffic pattern between an HMI and a PLC. Thus, the intrusion detection approach is very sensitive and is able to flag anomalies such as a message appearing out of position in the normal sequence or a message referring to a single unexpected bit.

The intrusion detection approach was tested on a production Modbus system that controls electric power supply at Tel Aviv University. The testing used more than 120 h of live traffic collected in two sessions several months apart. Despite its high sensitivity, the intrusion detection system has a very low false positive rate—five of the seven PLCs tested yielded perfect matches of the model to traffic, without a single false alarm over 111 h of operation. The system successfully flagged real anomalies produced when technicians were troubleshooting the HMI system. Moreover, the system helped identify a PLC that was configured incorrectly.

2. Related work

Media coverage of cyber attacks such as Stuxnet [3] has emphasized the need for strong and reliable security mechanisms for SCADA systems. Several researchers have focused on intrusion detection approaches for SCADA systems. Yang et al. [23] employed an auto associative kernel regression model coupled with a statistical probability ratio test to match patterns in simulated SCADA systems. Their model uses predetermined features, representing network traffic and hardware operating statistics, for intrusion detection.

Tsang and Kwong [19] proposed a detection approach based on an unsupervised anomaly-learning model. They developed an ant colony clustering model based multi-agent decentralized intrusion detection system. Their approach has been shown to reduce data dimensionality while preserving model accuracy.

Naess et al. [15] have proposed the use of interval-based sensors, procedural-based sensors and misuse-based detectors. Interval-based sensors identify if parameter values and method invocation frequencies fall within their predefined ranges. Procedural-based sensors are embedded at the entry and exit points of applications to monitor their execution patterns. Misuse-based detectors are positioned within application code at locations where vulnerabilities are known to exist.

Gao et al. [8] have presented a neural network based intrusion detection system that monitors the physical behavior of control systems to detect artifacts of command and response injection denial-of-service attacks.

Digital Bond [7] has specified a set of Modbus/TCP Snort rules for intrusion detection. The set includes fourteen rules that are broadly divided into three groups: (i) unauthorized Modbus protocol use; (ii) Modbus protocol errors; and (iii) scanning. Our method successfully detects all the anomalies encoded in the Digital Bond Snort rules. However, during an evaluation using a production Modbus/TCP system, our method flagged real anomalies that the Snort rules were unable to catch.

Nai Fovino et al. [16] have presented a state-based intrusion detection system. Their approach uses explicit knowledge of a SCADA system to generate a system virtual image. The virtual image represents the PLCs and remote terminal units (RTUs) 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).

The approach closest to our method was proposed by Cheung et al. [4]. 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 [17]. Three model-based techniques characterize expected/acceptable system behavior according to the Modbus/TCP specification: (i) a protocol-level technique that verifies the Modbus/TCP specifications for individual fields and groups of dependent fields in Modbus/TCP messages; (ii) a communication pattern modeling technique based on Snort rules; and (iii) a learning model that describes the expected trends in the availability of servers and services. The appliance was integrated into a control system testbed at Sandia National Laboratories and tested under a multi-step attack scenario. Our approach is also model-based, but it goes much deeper into the Modbus/TCP specifications and captures inter-packet relationships. Thus, it is able to perform all the tests of the first two levels of the system of Cheung and colleagues, but with higher sensitivity and with minimal training.

In subsequent work, Valdes and Cheung [20,21] incorporated adaptive statistical learning methods in two anomaly detection techniques—pattern-based detection for communication patterns among hosts and flow-based detection for traffic patterns in individual flows. In addition, they developed a visualization tool that assists human analysts. More recently, Briesemeister et al. [2] integrated these intrusion detection technologies into the EMERALD event correlation framework [18].

Due to the lack of access to production industrial control systems, many researchers have used SCADA testbeds for experimental investigations of vulnerabilities and for validating security solutions [9–13,22]. In contrast, one of the important aspects of our work is that the intrusion detection approach is evaluated using real traffic from a production SCADA network.

3. Modbus over TCP/IP

Modbus has become a *de facto* standard for industrial control systems. Many Modbus systems implement the communications layer using TCP as described in the Modbus over TCP/IP specification [14]. The specification defines an embedding of Modbus packets in TCP segments and assigns TCP port number 502 to the Modbus protocol. To maintain compatibility with Modbus over serial lines, payloads are limited to at most 253 bytes. Fig. 1 presents the structure of Modbus protocol messages.

The Modbus protocol employs a simple master-slave communication mode. The master device initiates transactions (called queries) and the slaves respond by supplying the requested data to the master or by performing the action requested in the query. Only one device can be designated as the master (usually the HMI) while the remaining devices are slaves (usually PLCs that control devices such as I/O transducers and valves). A slave sends a response message for every query that is addressed to it individually. In heterogeneous networks comprising both Modbus/TCP devices and serial Modbus devices, a gateway or a bridge is often used to connect the serial line sub-network to the IP network. In this case, the destination IP address identifies the bridging device that chains all the devices in the sub-network. The Modbus header (MBAP) has four fields covering seven bytes (Fig. 1), two of which are relevant to our work:

- *Transaction identifier*: This is a two-byte integer that pairs the request and the response corresponding to a transaction. A unique transaction ID is created for the request message from the master, which the slave includes in its response.
- *Unit identifier*: This is a single-byte integer that identifies the Modbus slave associated with a transaction. This is relevant to a Modbus gateway that chains several slaves.

3.1. Modbus PDU

Each PLC provides an interface based on the Modbus data model. The data model comprises "coil" (single-bit) and "register" (16-bit) tables, each containing elements numbered 1…*n*. For each table, the data model allows up to 65,536 data items. Read and write operations associated with these items can access multiple consecutive data items. The Modbus PDU has two fields that refer to the data model:

- *Function code*: The function code is a single-byte integer in the range 1…127. The Modbus standard defines the meaning of nineteen of the 127 possible function codes. In our data sets, we witnessed the appearance of only four different function codes, three read function codes (1, 2 and 3) and one write function code (5).
- *Payload*: The payload field has a variable size that is limited to 252 bytes. It contains parameters that are specific to the function code. A read request payload has two fields, a reference number and bit/word count. The reference number field specifies the starting memory address for the read operation. The bit/word count field specifies the number of memory object units to be read. The payload of the corresponding response has two slightly different fields, byte count and data. The byte count specifies the length of the data in bytes. The data field contains the values of the memory objects that were read. In addition to memory references, the payload of a write message has fields that specify the values that are to be written.

A successful request execution is indicated by a slave returning a response packet that echoes the function code of the request, followed by the relevant data (e.g., the bytes read as a result of a read command). A failure is indicated by an exception response, a two-byte error value comprising the original function code from the request PDU with its most significant bit set to a logical one.

3.2. Modbus/TCP security properties

The Modbus protocol does not defend itself in any way against a rogue master that sends commands to slaves. Furthermore, Modbus does not have long-term session semantics – the protocol simply involves separate two-message query-response sequences. However, in all the examples we encountered, the Modbus connection between the master and a specific slave is embedded in a single long-lived TCP connection. Moreover, at least one PLC we tested (Unitronics Vision350) only accepts a single TCP connection at a time on port 502. Therefore, an attacker attempting to control an already-controlled PLC would need to either hijack the existing TCP connection [1] and inject spoofed packets into the stream, or reset the existing connection and create a new connection. PLCs that allow multiple concurrent connections on port 502 are susceptible to much simpler attacks.

4. Protocol modeling for intrusion detection

Our work is based on the assumptions that a domain-specific Modbus intrusion detection system can be much simpler

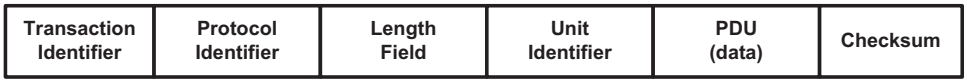


Fig. 1 – Modbus/TCP message structure.

than a general-purpose intrusion detection system and can have a much lower false positive rate. Due to the nature and purpose of SCADA systems, Modbus devices (e.g., HMIs and PLCs) are rarely added to or removed from an operating SCADA network. Furthermore, HMI-PLC communications are extremely regimented with few human-initiated actions. A key assumption is that the communications are highly periodic: the HMI repeatedly polls every PLC at a fixed frequency and issues a repeating sequence of commands. Thus, the traffic pattern allows simple models with extremely high predictive power that, in turn, enable the construction of intrusion detection systems with very low false positive error rates.

A preliminary inspection of our data sets yielded important observations that support the premises mentioned above. As we shall see later, the static nature of the SCADA system was validated by the near fixed number of SCADA network entities. Indeed, throughout the 120 h of traffic recorded over a period of five months, the production system comprised one HMI and six PLCs (five of which were active during the entire period). Furthermore, we observed that the HMI communicates separately with each of the PLCs. Each connection is maintained as a long-term TCP connection, which is immediately restarted upon disconnection. This behavior makes it possible to handle each PLC individually.

4.1. Using deterministic finite automata

Because SCADA systems have clear communication patterns, each HMI-PLC channel can be modeled as a deterministic finite automaton (DFA). A classical DFA is a five-tuple $(Q, \sum, \delta, q_0, F)$ comprising a finite set of states Q, a finite set of input symbols called the alphabet $\sum$, a transition function $\delta : Q \times \sum \rightarrow Q$, a start state $q_0 \in Q$ and a set of accept states $F \subseteq Q$.

Two adjustments are made in order to use a DFA to model Modbus data:

- No accept states are required because the intrusion detection system continuously monitors an endless repetitive stream. Instead, a Moore DFA, which associates an action with every state transition in δ, is employed. Any deviation from the predicted pattern triggers a δ transition with an associated error action that potentially raises an intrusion detection system alert depending on the severity of the deviation. Also, the start state is defined as the state corresponding to the first query recognized in the periodic traffic pattern (see Table 1).
- The Modbus features that identify a symbol in the alphabet $\sum$ must be selected. At an extreme, an overly naive alphabet with two symbols {*Query*, *Response*} could be used, which would expect a pattern of {*Query*, *Response*}*. However, as described later in this section, we incorporate much more granularity in the model by defining a symbol as the concatenation of several Modbus fields totaling 33 bits. State space explosion was not encountered despite using the much longer alphabet.

4.2. Channel separation and identification

The communication pattern for each PLC depends only on the HMI and is independent of the behavior of the other PLCs. Therefore, the recorded traffic was split into separate channels, each containing traffic for a single PLC. This facilitates the modeling and analysis of the behavior of each PLC separately without artificially increasing the state space of the model. This channel separation is easily done based on the IP address of a PLC.

A channel is defined by the tuple (*MasterIP*, *SlaveIP*) and is identified upon recognizing a Modbus packet (port 502 by default). If the master IP address is different from the (single) expected IP address, an alert "UNEXPECTED MASTER" is raised. Similarly, an alert is raised if the slave IP address is not an expected slave IP address. These conditions are equivalent to the Digital Bond Snort rules 1111006 and 1111007 [7].

As discussed in Section 3, some SCADA networks employ a Modbus gateway to chain several PLCs. In our production network, we observed PLC #5 functioning as a gateway that chained two PLCs. Communications between each of the chained PLCs and the HMI was independent (similar to the communications between an HMI and non-chained PLCs). Recall that the unit identifier field is used to address chained PLCs; thus, finer channel definition and separation are obtained using the three-tuple (*Master IP*,*Slave IP*,*Unit Identifier*). This definition enables each chained PLC to be treated individually, which, in our case, separates PLC #5.1 and PLC #5.2. Reference to a new unit identifier in a query message raises an alert. Note that the Digital Bond Snort rules do not catch such anomalies.

4.3. States and input symbols

Our basic observation is that the HMI-PLC traffic pattern for a PLC is periodic, i.e., the same sequence of queries and matching responses are repeated over and over. For example, our data showed that, in the case of PLC #2, the HMI sends a sequence of three fixed queries (and receives their matching responses) every 30 ms, and this pattern of six messages is maintained for many hours. Having identified the length of

Table 1 – Notation.

S_i	The ith state in the DFA.
s_i	The input symbol leading to S_i.
q_i	The ith query message in the sequence.
Q_i	The state reached after q_i (Q_1 is the state reached after the first query message in the sequence).
r_i	The ith response message in the sequence.
R_i	The state reached after r_i (R_1 is the state reached after the first response message in the sequence).

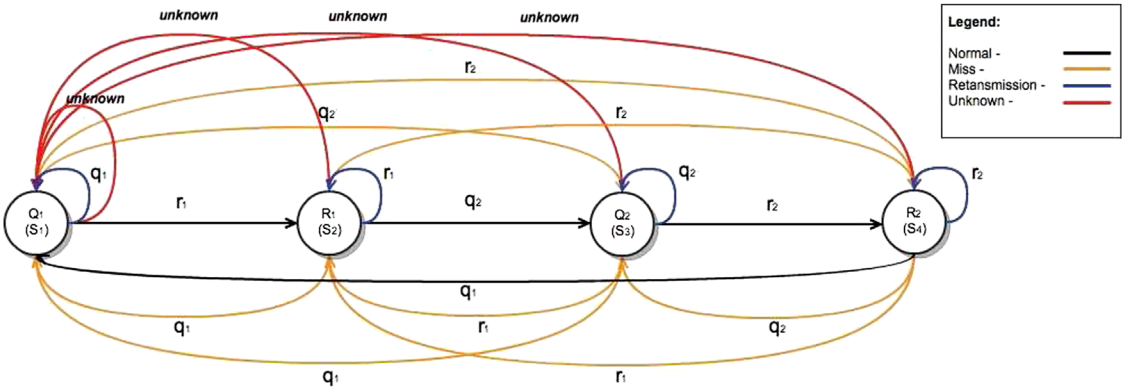


Fig. 2 – DFA representing a two-query Modbus traffic pattern. Normal transitions follow a periodic traffic pattern comprising two queries and their matching responses.

the pattern (six in the example), it is possible to define a DFA as shown in Fig. 2.

For each message in the pattern, we define a state and a "normal" transition. States that are reached after a query message are called Q-states. States that are reached after a response message are called R-states.

Recall that a Modbus query has the fields: transaction identifier (T.ID), function code (FC), reference number (RN), and bit/word count (this depends on the function code value; some function codes are followed by a bit count while others are followed by a word count). We define a symbol in the alphabet Σ as a four-tuple containing all these fields, except for T.ID. This yields 33-bit symbols (one bit for Q/R, eight bits for the function code, 16 bits for the reference number and eight bits for the bit/word count). Responses do not include the reference number, so the associated sixteen bits in the symbol are always zero.

Input symbols are categorized as "known" or "unknown." An input symbol is known if it was observed during the learning phase (described in Section 5) and has a matching DFA state. On the other hand, an input symbol is unknown if it does not have a matching DFA state or was not observed during the learning phase.

4.4. *Transition function*

A transition function in a Moore DFA is a transformation that for each (*Base State*, *Input Symbol*) tuple returns a (*Dest State*, *Operation*) tuple. The transition function implements the predicted behavior and expresses assumptions about network traffic characteristics by matching the correct state and operation to the given base state and input symbol.

Four types of transitions are defined. Note that S_i is the current state and s_j is the received input symbol:

- *Normal*: A normal transition occurs on a known symbol that leads to the next state in the periodic sequence, i.e., $s_j = s_{i+1}$.
  If the symbol triggering the normal transition is a query leading to a Q-state, the T.ID of the message is saved. If the symbol is a response, the T.ID of the current message is compared with the saved T.ID. If the T.IDs do not match, a T.ID mismatch counter is incremented. Only a handful of T.ID mismatches (less than 0.004% of the packets) were observed. All the mismatches were caused by dropped packets in the capture mechanism.
  As part of the normal transition, we implemented the in-packet validation tests suggested in [4,7], primarily by verifying that the packet payload length is at most 252 bytes. This mechanism flags buffer overflow attempts against the HMI (using fake responses from PLCs that are too long) or against the PLCs (using HMI queries that are too long). Note that it is not necessary to explicitly test the actual packet length against the in-packet count value, or a mismatch between the requested count and the supplied length in the response. This is because the count field is always part of the symbol, and any attempt to send too much or too little data would cause the packet to trigger an unknown transition (described below).
- *Retransmission*: A retransmission is an occurrence of a known symbol that is identical to the previous symbol, i.e., $s_j = s_i$.
  For such an occurrence, a self-loop is added to the DFA: *Dest State* = *Base State* = S_i. Retransmissions occur normally in TCP traffic due to momentary congestion, and they do not indicate a real anomaly in Modbus operations. Thus, the only action taken is to increment the retransmission counter.
- *Miss*: A miss is an occurrence of a known symbol s_j that appears in state S_i out of its expected position in the pattern, i.e., $s_j \neq s_{i+1}$.
  This typically occurs because the packet capture mechanism sometimes drops packets. Our view is that it is unlikely that the HMI would skip sending a packet in the normal pattern, and even more unlikely that the PLC would ignore a query. Therefore, a miss event is handled by a transition to the closest forward state (modulo *Pattern_Length*) that follows the normal s_j symbol. Again, because a miss is a relatively benign anomaly and most likely an artificial anomaly introduced by the packet capture mechanism, the only action performed is to increment the miss counter.
- *Unknown*: The most serious anomaly is the appearance of an unknown symbol. At worst, an unknown symbol indicates a malicious packet that has been injected into the TCP stream. However, other *a priori* interpretations are also possible. For example, an unknown query could

indicate human operator action, or it could indicate an unmodeled automatic response by the HMI to some condition observed in previous data, or it could indicate that the modeled pattern is too short to capture infrequent queries. An unknown response could also indicate the presence of a faulty PLC that responds with the wrong function code or the wrong amount of data. Regardless of the interpretation of the unknown symbol, a transition is made back to the first state (in the hope that the pattern will resynchronize), the unknown counter is incremented and an alarm is raised with the value of the symbol. Naturally, an unknown write operation is more serious than an unknown read operation.

Algorithm 1. Pattern modeling algorithm.

1. *Pattern_Length*←2
2. *DFA*←*DataLearning*(*Pattern_Length*)
3. *performance_value*←*ModelValidation*(*DFA*)
4. *while* (*performance_value* > *Threshold*) and
   (*Pattern_Length* < *Learning_Window_Size*) :
   (a) *Pattern_Length*←*Pattern_Length*+2
   (b) DFA←*DataLearning*(*Pattern_Length*)
   (c) performance_value←ModelValidation(DFA)
5. if (*Pattern_Length*≥*Learning_Window_Size*) :"FAILED"
6. else: return DFA

5. Creating the model

This section describes the process for creating the intrusion detection model.

5.1. Automatic model generation

A novel aspect of our approach is that the model can be learned automatically without any labeling of the training data. All that is required is a clean capture of normal traffic that is longer than the pattern.

The learning phase begins by capturing a fixed number of packets, indicated by *Learning_Window_Size*. Next, an inventory of the identified queries and responses in the window is taken, and several checks are performed on the inventory. The checks involve verifying that each query has a valid response and verifying that each response has a preceding query. Then, an iterative method is employed to create the smallest DFA that models the sniffed Modbus packets (Algorithm 1).

The iterations begin with an initial estimate of two for *Pattern_Length* (i.e., one query and one response, the shortest possible legitimate pattern). In each iteration, the current pattern candidate is defined as the first *Pattern_Length* Modbus messages (starting with a query message) in the window. From this candidate pattern, a DFA is constructed as described in Sections 4.3 and 4.4. Then, the created DFA is run against *Validation_Window_Size* captured Modbus messages, and the numbers of misses, retransmissions and unknowns are counted (this window is assumed to have no unexpected network events or activities, i.e., no anomalies). Based on the counter values, a performance value P is defined as:

$$P = \frac{normal}{total}. \tag{1}$$

If P is below a set threshold, then *Pattern_Length* is too small; so it is incremented by two and a new iteration is started. If *Pattern_Length* exceeds *Learning_Window_Size*, the iterations terminate with failure.

5.2. Setting the threshold

Each channel, and thus each PLC, is characterized by its own periodic pattern length. We denote the periodic pattern length as k and the candidate pattern length as n. For each channel, the value of k must be discovered separately. The performance threshold in Algorithm 1 should be defined such that it differentiates the correct pattern length from other shorter/longer candidate pattern lengths. Manually tuning the threshold to a good value is a challenging task. A better choice is to have a self-tuning threshold.

We were able to analytically define a threshold that accomplishes the desired differentiation. Two assumptions were made: (i) a clean validation window of length V exists (for simplicity, we assume that the validation window size obeys $V \bmod k = 0$ and that $V \to \infty$); and (ii) the pattern consists of k distinct messages.

Given a periodic sequence of distinct messages of length k and a clean validation window of size *Validation_Window_Size* $= V$, a DFA with size corresponding to the candidate length n is constructed. For each candidate length, the DFA performance is evaluated based on the validation window.

We consider three cases:

- The candidate length is shorter than the actual pattern length, i.e., $n \le k$. Then, the model mistakenly recognizes the last $k-n$ messages in the periodic sequence as unknown. Thus, for each appearance of the periodic pattern, the model counts n normal transitions and $k-n$ unknown transitions (corresponding to the unknown messages), resulting in $P = n/k$.
- The candidate length is a multiple of the actual pattern length, i.e., $n = i \cdot k$ where $i \in N$. The model contains multiple repetitions of the complete periodic sequence. Thus, no unknowns occur because the DFA contains all the messages that appear in the validation window. Furthermore, since the DFA contains an exact multiple of the periodic sequence, no misses or retransmissions occur. Consequently, $P = 1$.
- The candidate length is longer than the actual pattern length but is not a multiple of the actual pattern length. Thus, $n = i \cdot k + r$ for $1 \le r < k$. The first $i \cdot k + r$ symbols trigger normal transitions. Then, the DFA expects symbol s_1 but encounters symbol s_{r+1}, causing a miss. However, since a miss transition has a next state that is the closest forward state matching the input, the DFA transitions to state S_{r+1}, effectively resynchronizing the expected pattern with the input. All subsequent symbols trigger normal transitions until s_{2ik+r} triggers another miss, and so

on. In every block of $i \cdot k$ input symbols (except for the first one), the DFA triggers a single miss and $i \cdot k-1$ normal transitions. Thus, when $V \to \infty$, we obtain $P = (i \cdot k-1)/i \cdot k$. Note that the performance value is independent of r, and the same $P = (i \cdot k-1)/i \cdot k$ is obtained for all values $n = i \cdot k + 1, \ldots, i \cdot k + (k-1)$.

In summary, if the pattern comprises k distinct symbols, the input is perfectly clean and the validation window $V \to \infty$, then the performance value is given by the following function of candidate length n:

$$P = \begin{cases} \frac{n}{k} & n \le k \\ 1 & n = i \cdot k, \quad i \ge 1 \\ \frac{i \cdot k-1}{i \cdot k} & i \cdot k + 1 \le n \le i \cdot k + (k-1), \quad i \ge 1 \end{cases} \tag{2}$$

The threshold T must be tuned so that it causes Algorithm 1 to terminate for $n=k$. To achieve this, it is necessary to set $T = (k-1)/k$, except that k is unknown. However, as long as $k \le n$, the sequence $P(n)$ is increasing, so it suffices to set T high enough so that $n = k-1$ is not accepted, i.e., setting $T = n/(n+1)$ and having Algorithm 1 terminate when $P > T$ is enough. Note that the threshold provides less discrimination as n increases because the value $n/(n+1)$ moves closer to one.

6. Data acquisition and preliminary analysis

Due to the proprietary nature and potential sensitivity of SCADA operations, real SCADA network data is rarely released to researchers. In fact, we are unaware of any publicly accessible Modbus/TCP data sets that contain more than a handful of packets. Therefore, most researchers rely on data sets extracted from SCADA testbeds.

An important aspect of this work is that we were able to collect and analyze long traces from a production Modbus network. We discovered that the facility manager at our university (Tel Aviv University) uses a Modbus/TCP-based system to monitor the campus power grid, and that the system uses the campus-wide IP network for communications. With the assistance of the university CISO, we were able to tap into the Modbus communications and record the traffic during two time periods, producing two data sets described in Table 2.

One of the research goals was to keep the network dependency of our method as low as possible by not using any prior knowledge about the network. Therefore, a preliminary network analysis was conducted to provide basic insights. The analysis focused on gathering information about SCADA entity identification and traffic statistics. The analysis was performed using WireShark and automated scripts written in Python that employ Impacket [5] and Pcapy [6] modules for network packet handling. After performing the analysis, we met with the facility manager to validate the findings and obtained the vendor and model names of the system components.

In Data Set #1, we observed four Modicon PLCs and one Satec PLC (with two chained unit IDs), all controlled by an Afcon Pulse HMI. In Data Set #2, we observed the same PLCs along with one additional Modicon PLC.

Using a splitting procedure written in Python, the primary data files were divided into sub-files, each containing packets for a given time frame. Data Set #1 was split into 630 time frames, each 10 MB in size (equivalent to two minutes of traffic). Data Set #2 was split into 1340 time-frames, each 26 MB in size (equivalent to five minutes of traffic). These time frames were used as basic units for calculations and comparisons in our experimental analysis.

7. Model validation

To validate the DFA-based model, we implemented the DFA construction method in Python using Impacket and Pcapy modules for network packet handling. The analysis results and the suspected anomalous traffic were verified and validated against the network activity log with the facility manager.

7.1. Model creation with automatic threshold tuning

Running Algorithm 1 with an auto-tuned threshold on the data sets with *Learning_Window_Size*=50 and *Validation_Window_Size*=100 yielded very good results. In particular, the periodic pattern lengths were accurately identified for all the PLCs in the two data sets. Fig. 3 shows the performance of the method for PLC #1. In the case of Data Set #1, the method successfully identified the periodic pattern length $k=16$. Note the local maxima at $n=16$ and 32.

7.2. Basic model validation

Basic model validation focused on the ability of the model to represent normal network traffic using a DFA. Two parameters were used to measure DFA quality: *Pattern_Length*, and the unknown, miss and retransmission rates. Recall that *Pattern_Length* is the smallest integer whose performance passes the threshold. Successfully fitting a *Pattern_Length* that makes a DFA pass the performance threshold demonstrates that the DFA represents the traffic captured in the validation window accurately. Clearly, *Pattern_Length*=*Validation_Window_Size* successfully passes the threshold.

Table 3 summarizes the *Pattern_Length* results for the PLCs. For every PLC, our method successfully constructed a DFA representing very short periodic patterns. For example, in the

Table 2 – Modbus data sets.

Data set	Start date	End date	Duration (hours)	File size (GB)
#1	16.1.12 17:40	17.1.12 13:50	20	6.3
#2	19.6.12 9:00	24.6.12 00:50	111	35.5

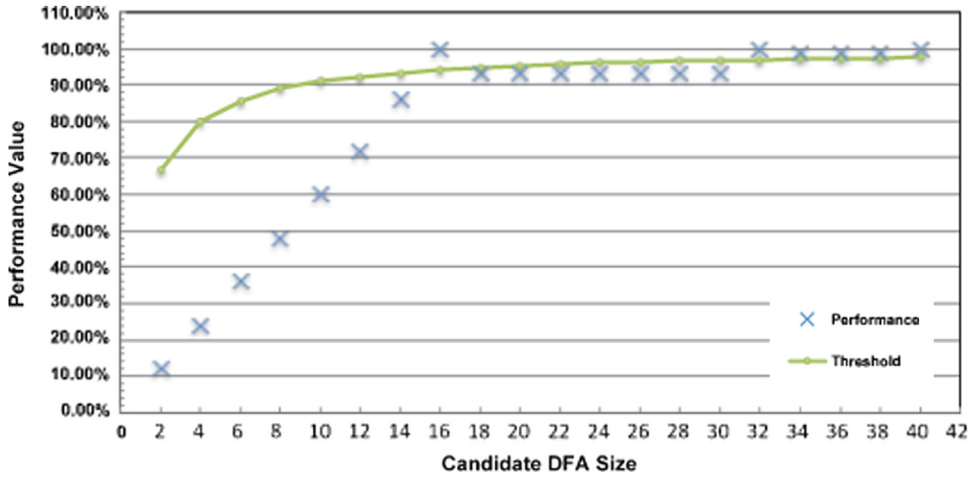


Fig. 3 – Performance vs. candidate DFA size for PLC #1 using Data Set #1.

Table 3 – *Pattern_Length* results for the PLCs recognized in Data Sets #1 and #2.

Data set #1		Data set #2	
PLC	$Pattern_{Length}$	PLC	$Pattern_{Length}$
#1	16	#1	18
#2	6	#2	4
#3	6	#3	6
#4	6	#4	6
#5.1	2	#5.1	2
#5.2	2	#5.2	2
#6	–	#6	6

case of Data Set #1, the largest DFA was obtained for PLC #1 (*Pattern_Length*=16). Hence, the network traffic between PLC #1 and the HMI has a periodic pattern comprising eight queries and their matching responses.

The second parameter for measuring model quality consists of the unknown, miss and retransmission rates. Fig. 4 (left column) shows that, except for a few distinct peaks throughout the entire model run, all three anomaly counters have extremely low values and represent only verified network congestion and packet drops. In fact, 625 of the 630 time frames in Data Set #1 are "quiet" – the unknown counter has a value of zero, and the miss and retransmission counters are less than fifteen for each time frame. This validation was performed for every PLC with similar results.

7.3. Anomaly detection

In Data Set #1, the unknown rate is very low for all the PLCs – at most 0.39% of the packets. However, the unknown symbols are not evenly distributed over time. In fact, 97.7% of the time frames in Data Set #1 are completely quiet. Fig. 4 (left column) clearly shows two interesting periods of anomalous activity in Data Set #1 near time frames #84 and #460. Furthermore, Fig. 4 (right column) shows that these events affect all the PLCs at the same time, making them even more suspicious. As discussed in Section 7.4, these are not false positives, but actual anomalies flagged by the system.

Correlating anomalous activity observed in multiple devices is an important aspect of intrusion detection. An intrusion detection system based on our approach can provide valuable input feed to an event correlation system such as EMERALD [18] or a commercial system such as the HP ArcSight Security Intelligence platform.

Recall that the DFA transition function is defined such that, after each unknown input symbol, the DFA state is changed to the start state. Due to the arbitrary position of an unknown symbol in the periodic sequence, the next transition is likely to be a miss or a retransmission. Therefore, the three counters are technically correlated due to the manner in which the model was constructed. Fig. 4 (left column) clearly shows this correlation, with obvious spikes in all three counters near time frames #84 and #460.

7.4. Real anomalies

After analyzing the network using our modeling method, we met with the facility manager and examined the suspicious messages and events versus the network logs. All the unknown transitions were verified to be indeed suspicious and not false alarms. The prominent interrupts in Fig. 4 were found to be caused by technicians who were troubleshooting problems with the system that day.

8. Results for data set #2

Data Set #2 was collected five months after Data Set #1. During the five months, the SCADA system was upgraded by technicians. The upgrades caused several significant changes to the SCADA network traffic. First, a new PLC appeared in addition to the five encountered in Data Set #1. The new PLC, a Modicon PLC similar to PLCs #2, #3 and #4, is labeled as PLC #6. Table 3 shows the changes observed in the traffic patterns of two PLCs. The *Pattern_Length* of PLC #1 increased to 18, adding an additional query and its matching response to the periodic sequence. Also, PLC #2 dropped a query and its matching response, resulting in a shorter *Pattern_Length*=4.

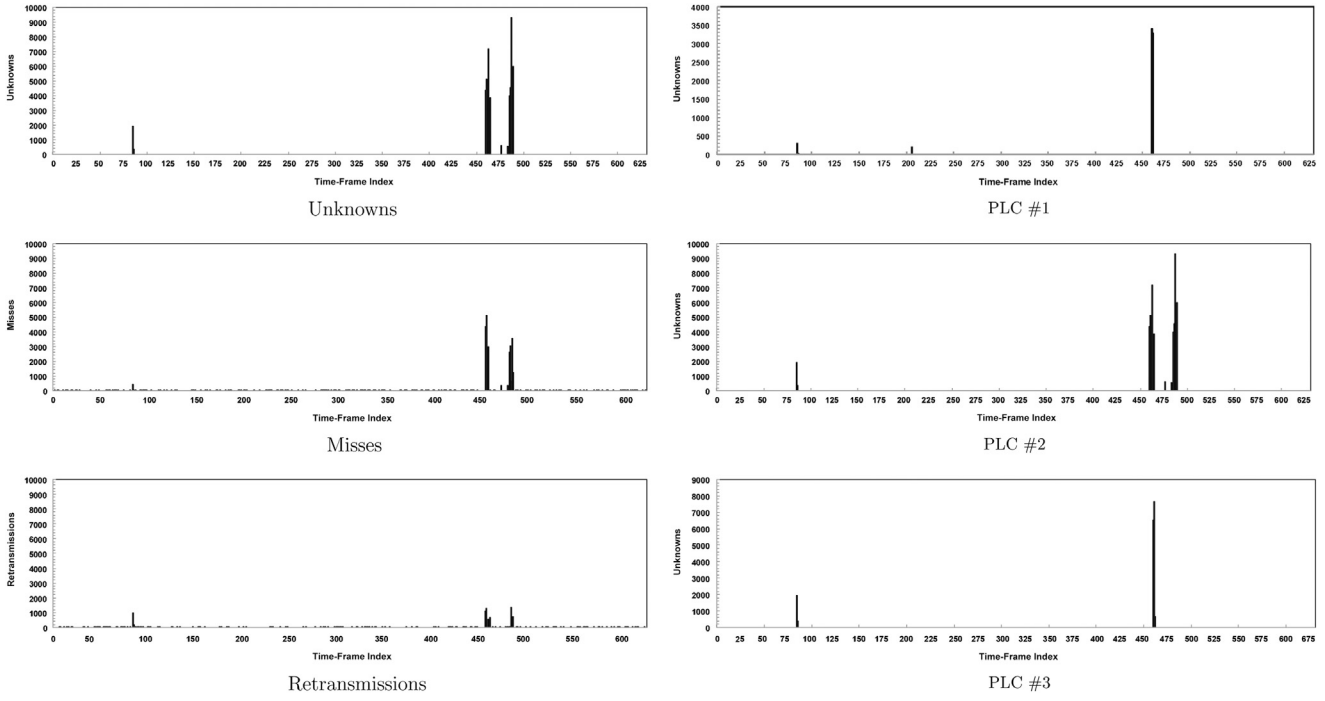


Fig. 4 – Correlation between event types in PLC #2 for Data Set #1 (left column); Correlation between unknowns across different PLCs (right column).

Distinctly different performance statistics were obtained for Data Set #2. Specifically, two different performance effects were observed compared with Data Set #1.

The first effect was the perfect modeling of network traffic for five of the seven PLCs. For each PLC, our method modeled the network traffic perfectly for 111 h without any unknown messages. Thus, for these PLCs, it can be concluded that, despite the high sensitivity of the DFA approach, no false alarms were raised.

The second effect, concerning only PLC #1, was the significant increase in the frequency of misses and unknowns—from 0.09% in Data Set #1 to 0.4% in Data Set #2. Even more problematic is fact that the percentage of quiet time frames dropped to only 66% (see Table 5). In other words, the unknown events were not localized to a few anomalous time frames as in Data Set #1, but were spread throughout the data set. A closer examination of the unknown events versus time revealed that the control of PLC#1 in Data Set #2 operated with three separate time periods. Aside from the high frequency pattern that is well modeled by the DFA, we observed two other periodic patterns in Data Set #2 that are much slower: a low frequency periodic pattern with a period $T_1 = 24$ h observed four times, and a mid frequency periodic pattern with a period $T_2 = 15$ min observed 446 times. Because we used a five-minute time frame for Data Set #2, the 15-min pattern produced the effect that only 66% of time frames were quiet, i.e., the extra messages in the pattern occurred in one out of every three time frames. Our facility manager verified that both patterns were normal, noting that the daily periodic pattern ($T_1 = 24$ h) was for resetting various PLC counters, and the quarter-hourly pattern ($T_2 = 15$ min) was for averaging a set of control process counters.

9. Modeling multi-period traffic patterns

Modeling SCADA traffic that has multiple time periods is a challenge for our approach. Using the DFA approach naively would require capturing more than $T_1 = 24$ h of traffic just to construct the model. In addition to taking a long time, this would also produce a very large DFA with approximately 9.6 million states. The DFA would also be inaccurate because it is difficult to capture so much clean traffic. Consequently, we devised a multi-DFA method for modeling the traffic.

A multi-DFA is a concatenation of several DFAs, each implementing our method. As shown in Fig. 5, the proposed multi-DFA model comprises two serially-connected DFAs. The Level-1 DFA corresponds to the fast periodic sequence and the Level-2 DFA corresponds to the slow periodic sequence. A new input symbol is passed to the Level-1 DFA. If the Level-1 DFA marks the symbol as unknown, no alert is raised (unlike the single DFA case), but the symbol is passed to the Level-2 DFA. If the Level-2 DFA recognizes it as a known symbol, then no alert is raised. Otherwise, the Level-2 DFA marks the symbol as unknown and raises an alert.

9.1. *Injecting time-difference symbols*

Unlike the constant-rate input stream received by the Level-1 DFA, the Level-2 DFA receives bursts of input comprising symbols marked as unknown from the main traffic stream processed by the Level-1 DFA. When the SCADA traffic has a second slow pattern, the quiet times for the Level-2 DFA are expected to appear periodically. This means that the Level-2 DFA may experience long periods of inactivity when traffic is matched by the Level-1 DFA. Indeed, a deviation from the

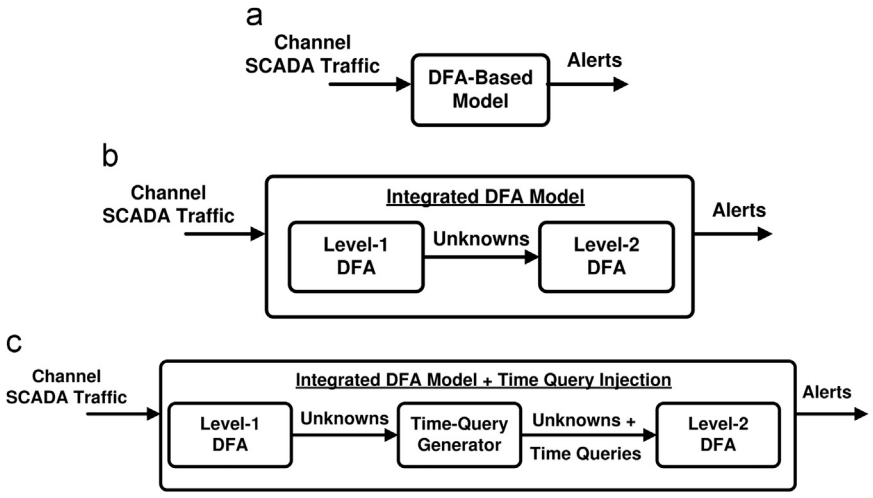


Fig. 5 – Model interfaces. (a) Single DFA, (b) integrated DFA, and (c) integrated DFA with time-query injection mechanism.

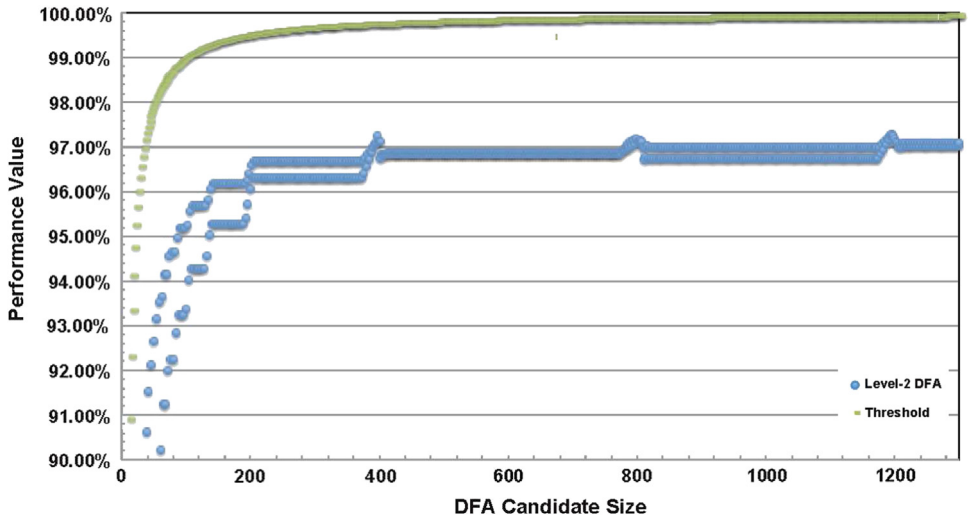


Fig. 6 – Performance vs. candidate Level-2 DFA size for PLC#1 on residual data in Data Set #2.

expected duration of the quiet period should trigger anomaly detection by the Level-2 DFA.

In order to add timing information to the model and still keep its structure and behavior, we decided to manipulate the model by injecting synthetically generated symbols into the Level-2 DFA input stream. These symbols were generated by a thread that measured the time between the arrival of unknown symbols.

The thread assumes that the current unknown symbol arrives ΔT after the previous unknown symbol. Then, a synthetic time-difference symbol (Q, tdiff, RN, Count) is generated using an unused Modbus function code for tdiff. We use the reference number (RN) field in the symbol to indicate the time scale (RN=1 for s and RN=60 for min), and Count to indicate the actual time difference in the appropriate units. Thus, when ΔT is in microseconds, the symbol is generated as follows:

$$\begin{cases} \text{no symbol is injected} & \text{if } \Delta T < 1\,\text{s} \\ \text{symbol} = \left(\text{Q}, \text{tdiff}, 1, \lfloor \frac{\Delta T}{10^6} \rfloor\right) & \text{if } 1\,\text{s} < \Delta T < 1\,\text{min} \\ \text{symbol} = \left(\text{Q}, \text{tdiff}, 60, \lfloor \frac{\Delta T}{6 \cdot 10^7} \rfloor\right) & \text{if } \Delta T > 1\,\text{min} \end{cases}$$

In other words, a quiet time under one second is ignored, a quite time between 1 s and 1 min is counted in seconds, and a quiet time over 1 min is rounded to minutes.

9.2. *Constructing the level-2 DFA*

Given the residual stream of unknowns from the Level-1 DFA, including the injected time symbols, it is necessary to identify the length of the Level-2 pattern in order to construct its DFA. Fig. 6 shows that our auto-tuning method performs poorly on the residual data sent to the Level-2 DFA by the Level-1 DFA. In fact, although the performance graph has a similar shape to the graphs in Section 7.1, the analytic threshold $T = n/(n+1)$ is consistently too high and does not permit the identification of the correct period.

One reason for the failure is the repeating states in the periodic sequence. Our analysis in Section 5.2 was based on the assumption of distinct queries. However, the residual symbol stream pattern (the slow pattern) contains a prominent repetition (up to 90 times) of two queries and their responses. This invalidates our analysis.

Another reason is that the pattern length is much longer. The slow pattern comprises approximately 200 queries and their responses. The large candidate pattern lengths make the difference between the performance P (see Eq. (1)) for correct and incorrect lengths smaller and more sensitive to errors.

The third reason is an inconsistent pattern. Unlike the fast pattern, the slow pattern exhibits some inconsistency in its length as well as in its symbols. Specifically, the pattern length varies across different instances, averaging a value of 400; the pattern symbols vary slightly between instances of the sequence.

For the reasons mentioned above, we had to use a different method to identify the pattern length. Instead of a threshold, we chose the DFA size by evaluating DFA performance for all pattern lengths between 2 and 1,300 and selected the length with the maximal performance value. Table 4 shows the values chosen for the pattern length.

Table 4 – *Pattern_Length* results for PLC #1 with Data Set #2.

Model type	*Pattern_Length*
Single DFA	8
2-Level DFA	418=18 + 400

9.3. *Two-level DFA model performance*

The first and most obvious performance benefit of the hierarchical two-level model is a relatively small DFA (i.e., it is necessary to handle only 418 states in total). Second, as shown in Table 5, the two-level model improves the unknown rate dramatically. In particular, the unknown rate drops by two orders of magnitude from 0.4% to 0.004%. In 110 h of captured traffic containing more than 40 million packets, the raw number of unknowns dropped from 176,000 with a single-level DFA to only 1982 packets with a two-level DFA. The fraction of quiet time frames also improved, increasing to 77%. A closer examination of the time frames revealed that many of them are only slightly "dirty," meaning that the unknown counter values in the time frames was under five. (Note that in Table 5 an "almost quiet" time frame is one with an unknown counter value that is less than five.)

Table 5 – Model performance for PLC #1 with Data Set #2.

Model type	Normal (%)	Miss (%)	Retransmission (%)	Unknown (%)
1-Level	99.49	0.10	0.0007	0.4
2-Levels	99.88	0.10	0.0007	0.0045
Model type	**Quiet time frame (%)**			**Almost quiet time frame (%)**
1-Level	66.2687			66.2687
2-Levels	76.8657			99.5522

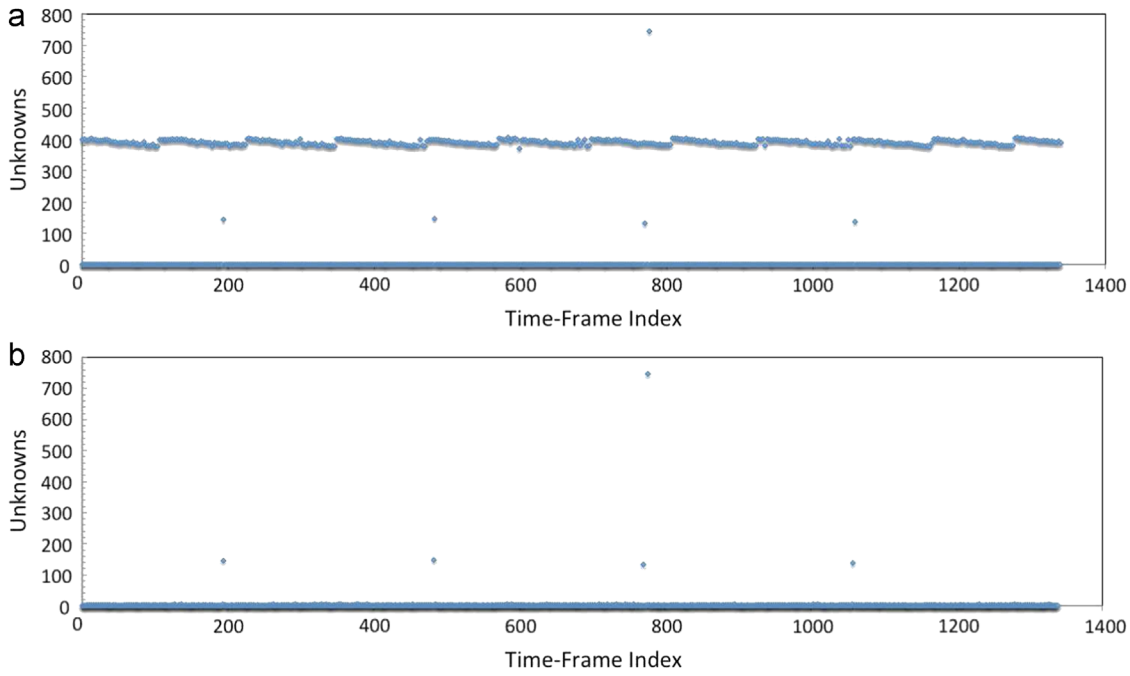


Fig. 7 – Unknown counter value vs. time frame index for PLC #1 with Data Set #2. (a) Simple DFA model and (b) Integrated DFA and time-query injection model.

We believe that these unknowns are due to the inconsistent nature of the slow pattern, and are not related to intrusions or any other abnormal network activity.

Fig. 7(b) shows that the Level-2 DFA models the slow periodic pattern ($T_2 = 15$ min) well. The unknown counter values are lower in all the time frames compared with the counter values for the simple DFA model in 7(a). The very slow periodic pattern ($T_1 = 24$ h) is still observed four times in Data Set #2.

10. Conclusions

The DFA-based approach, which is designed specifically for intrusion detection in SCADA networks, has two promising characteristics. First, it exhibits very low false positive rates despite its high sensitivity. Second, it can flag real anomalies that are missed by the Snort rules specified by Digital Bond [7].

The single-DFA intrusion detection model handles single-period traffic patterns very well. However, the performance degrades for multi-period traffic patterns—the slower patterns increase the false positive rate. Experiments demonstrate that the hierarchical multi-DFA extension can handle multi-period traffic patterns well. However, the multi-DFA model requires additional work to achieve performance that is comparable with that obtained for single-period traffic. Our future research will investigate the multi-DFA model thoroughly and will develop tuning strategies to ensure that it consistently exhibits superior performance.

Evaluating an intrusion detection system using live traffic from a production SCADA system provides valuable insights. But the approach has two inherent limitations, which we will address in our future research. First, we did not attempt to inject malicious traffic into the network to avoid interfering with the SCADA system; we will test such aggressive scenarios in a laboratory environment. Second, the approach was only applied to a single Modbus/TCP system; to further validate our results, we will test the approach on other Modbus/TCP systems.

Acknowledgments

We thank Eli Barnea for encouraging us to check if our campus facility management system uses Modbus/TCP. We also thank Yacov Menachem for showing us how the production SCADA system was organized and for helping map our observations to real events. Finally, we thank Ariel Biener for his assistance in capturing the Modbus traffic used in our research.

REFERENCES

[1] S. Bellovin, Security problems in the TCP/IP protocol suite, ACM SIGCOMM Computer Communication Review 19 (2) (1989) 32–48.

[2] L. Briesemeister, S. Cheung, U. Lindqvist, A. Valdes, Detection, correlation and visualization of attacks against critical infrastructure systems, in: Proceedings of the Eighth International Conference on Privacy, Security and Trust, 2010, pp. 17–19.

[3] T. Chen, Stuxnet, the real start of cyber warfare?, IEEE Network 24 (6) (2010) 2–3.

[4] S. Cheung, B. Dutertre, M. Fong, U. Lindqvist, K. Skinner, A. Valdes, Using model-based intrusion detection for SCADA networks, in: Proceedings of the SCADA Security Scientific Symposium, 2007, pp. 127–134.

[5] Core Security Technologies, What is Impacket? Buenos Aires, Argentina, ⟨oss.coresecurity.com/projects/impacket.html⟩, 2003.

[6] Core Security Technologies, What is Pcapy? Buenos Aires, Argentina, ⟨oss.coresecurity.com/projects/pcapy.html⟩, 2010.

[7] Digital Bond, Modbus TCP Rules, Sunrise, Florida, ⟨www.digitalbond.com/tools/quickdraw/modbus-tcp-rules⟩.

[8] W. Gao, T. Morris, B. Reaves, D. Richey, On SCADA control system command and response injection and intrusion detection, presented at the *eCrime Researchers Summit*, 2010.

[9] B. Genge, C. Siaterlis, I. Nai Fovino, M. Masera, A cyber-physical experimentation environment for the security analysis of networked industrial control systems, Computers and Electrical Engineering 38 (5) (2012) 1146–1161.

[10] A. Giani, G. Karsai, T. Roosta, A. Shah, B. Sinopoli, J. Wiley, A testbed for secure and robust SCADA systems, ACM SIGBED Review 5 (2) (2008) 4 Article no. 4.

[11] A. Hahn, B. Kregel, M. Govindarasu, J. Fitzpatrick, R. Adnan, S. Sridhar, M. Higdon, Development of the PowerCyber SCADA security testbed, in: Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research, article no. 21, 2010.

[12] D. Kang, H. Kim, Development of a testbed and security devices for SCADA communications in an electric power system, in: Proceedings of the Thirty-First International Telecommunications Energy Conference, 2009.

[13] M. Mallouhi, Y. Al-Nashif, D. Cox, T. Chadaga, S. Hariri, A testbed for analyzing security of SCADA control systems (TASSCS), in: Proceedings of the IEEE Innovative Smart Grid Technologies Conference, 2011.

[14] Modbus-IDA, Modbus Messaging on TCP/IP Implementation Guide V1.0b, Hopkinton, Massachusetts ⟨www.modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf⟩, 2006.

[15] E. Naess, D. Frincke, A. McKinnon, D. Bakken, Configurable middleware-level intrusion detection for embedded systems, in: Proceedings of the Twenty-Fifth IEEE International Conference on Distributed Computing Systems, 2005, pp. 144–151.

[16] I. Nai Fovino, A. Carcano, T. De Lacheze Murel, A. Trombetta, M. Masera, Modbus/DNP3 state-based intrusion detection system, in: Proceedings of the Twenty-Fourth IEEE International Conference on Advanced Information Networking and Applications, 2010, pp. 729–736.

[17] M. Roesch, Snort – Lightweight intrusion detection for networks, in: Proceedings of the Thirteenth USENIX Conference on System Administration, 1999, pp. 226–238.

[18] SRI International, Event Monitoring Enabling Responses to Anomalous Live Disturbances (EMERALD), Menlo Park, California ⟨www.csl.sri.com/projects/emerald⟩.

[19] C. Tsang, S. Kwong, Multi-agent intrusion detection system for an industrial network using ant colony clustering approach and unsupervised feature extraction, in: Proceedings of the IEEE International Conference on Industrial Technology, 2005, pp. 51–56.

[20] A. Valdes, S. Cheung, Communication pattern anomaly detection in process control systems, in: Proceedings of the IEEE Conference on Technologies for Homeland Security, 2009, pp. 22–29.

[21] A. Valdes, S. Cheung, Intrusion monitoring in process control systems, in: Proceedings of the Forty-Second Hawaii International Conference on System Sciences, 2009.

[22] C. Wang, L. Fang, Y. Dai, A simulation environment for SCADA security analysis and assessment, in: Proceedings of the International Conference on Measuring Technology and Mechatronics, vol. 1, 2010, pp. 342–347.

[23] D. Yang, A. Usynin, J. Hines, Anomaly-based intrusion detection for SCADA systems, in: Proceedings of the Fifth International Topical Meeting on Nuclear Plant Instrumentation, Control and Human–Machine Interface Technologies, 2006, pp. 12–16.