Accurate Modeling of Modbus/TCP for Intrusion Detection in SCADA Systems

(Extended abstract)

Niv Goldenberg and Avishai Wool,

School of Electrical Engineering, Tel Aviv University

{nivg99@gmail.com, yash@acm.org}

January 4, 2013

Abstract

Modbus/TCP is used in SCADA networks to communicate between the Human Machine Interface (HMI) and the Programmable Logic Controllers (PLCs). Therefore, deploying Intrusion Detection Systems (IDS) on Modbus networks is an important security measure. In this paper we introduce a model-based IDS specifically built for Modbus/TCP. Our approach is based on a key observation: Modbus traffic to and from a specific PLC is highly periodic. As a result, we can model each HMI-PLC channel by its own unique deterministic finite automaton (DFA). Our IDS looks deep into the Modbus packets and produces a very detailed model of the traffic. Thus, our method 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. We designed an algorithm to automatically construct the channel's DFA based on about 100 captured messages. A significant contribution is that we tested our approach on a production Modbus system. Despite its high sensitivity, the system enjoyed a super-low false-positive rate: on 5 out of the 7 PLCs we observed a perfect match of the model to the traffic, without a single false alarm for 111 hours. Further, our system successfully flagged real anomalies that were caused by technicians troubleshooting the HMI system—and the system also helped uncover one incorrectly configured PLC.

1 Introduction

1.1 Background

Supervisory Control and Data Acquisition (SCADA) networks are used for monitoring various industrial, infrastructure and facility-based processes. Some SCADA systems are used in critical national infrastructures including chemical plants, electric power generation, transmission and distribution, water distribution networks, and waste treatment. Such SCADA systems have a strategic significance due to the great damage consequences of any fault or malfunction.

SCADA systems typically incorporate sensors and actuators controlled by Programmable Logic Controllers (PLCs), and a Human Machine Interface (HMI). SCADA systems were originally designed for serial-line communications, and were built on the premise that all entities operating in the network are legitimately installed, perform the intended logic and follow the protocol's pattern. The result is that many such systems have almost no measures of defense against deliberate attacks. Network components do not verify the identity and permissions of other components associated with them (Authentication and Authorization), nor do they verify the messages' content and legitimacy (Data Integrity), and all the data transferred over the network is plaintext, without any encryption.

However, economic trends drive SCADA technology away from serial lines toward off-the-shelf components and IP communication protocols. In particular Modbus/TCP is commonly used in SCADA networks to communicate between the HMI and the PLCs. If attackers would be able to inject Modbus messages into the network they could cause significant damage. Therefore, deploying Intrusion Detection Systems (IDS) on Modbus networks is an important security measure.

1.2 Related Work

The domain of SCADA-specific Intrusion Detection Systems (IDS) is fairly active. Media attention to cyber attacks such as Stuxnet (cf. Chen [14]) has emphasized the need for reliable IDSs.

Several different approaches for securing SCADA networks have been published. Yang et al. [29] use the Auto Associative Kernel Regression (AAKR) model coupled with the Statistical Probability Ratio test (SPRT) and apply them to local simulated SCADA systems looking for matching patterns. The AAKR model uses predetermined features, representing network traffic and hardware-operating statistics.

An approach featuring an unsupervised anomaly-learning model was proposed by Tsang and Kwong [26]. Using an Ant Colony Clustering Model (ACCM) based multi-agent decentralized IDS they managed to reduce data dimensions while keeping model accuracy.

Næss et al. [24] use interval-based, procedural-based IDS sensors and misuse-based IDS detectors. Interval-based sensors are responsible for identifying whether parameter values and method invocation frequencies fall within their predefined ranges. Procedural-based sensors embedded at the entry or exit points of application monitor its execution patterns. Misuse-based detectors reside within the application's source code at those locations where known vulnerabilities exist.

Gao et al. [18] present a neural network based intrusion detection system which monitors control system physical behavior to detect artifacts of command and response injection denial of service (DOS) attacks.

Digital Bond Inc. [2] has developed a set of Modbus/TCP Snort rules, as part of its SCADA IDS research project. The set consists of 14 rules that can be broadly grouped to three groups: Unauthorized Modbus use, Modbus protocol errors and Scanning. Our method successfully detects all the anomalies encoded into the Snort rules. However, in our evaluation on a production Modbus/TCP system, our method flagged real anomalies that Digital Bond Snort rules are unable to catch.

Fovino et al. [17] present a State-Based Intrusion Detection System. In their approach, explicit knowledge of the SCADA system is used to generate a System Virtual Image (SVI). The SVI represents the PLCs and RTUs of the monitored system, with all their memory registers, coils, inputs and outputs. The SVI is kept updated by using a periodic active synchronization procedure and by a feed generated of the conventional packet analysis performed by the IDS (searching for known signatures).

The approach closest to ours was introduced by Cheung et al [15]. They designed a multi-algorithm IDS appliance for Modbus/TCP, containing pattern anomaly recognition, Bayes analysis of TCP headers, and stateful protocol monitoring complemented with customized Snort rules [25]. Three model-based techniques characterize the expected/acceptable system behavior according to the Modbus/TCP specification: The protocol-level technique verifies Modbus/TCP specifications for individual fields and for groups of dependent fields in the Modbus/TCP messages; The communication patterns modeling technique, is based on Snort rules; and a learning model that describes the expected trends in the availability of servers and services. The appliance was integrated into a control system testbed implemented at Sandia National Laboratories and experimented on a multistep 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 Cheung et al.'s system–with higher sensitivity and with minimal training,

In subsequent work, Valdes and Cheung [27, 28] incorporated adaptive statistical learning methods for two anomaly detection techniques, namely, pattern-based detection for communication patterns among hosts, and flow-based detection for traffic patterns for individual flows. In addition, they developed a visualization tool that assists human analysts. Most recently [13], they integrated the developed intrusion detection technologies into the EMERALD [10] event correlation framework.

Due to lack of access to production Process Control System (PCS) networks, many works deal with the issue of building a SCADA testbed that enables experimental capabilities of checking vulnerabilities and validating security solutions [19, 16, 23, 20, 22, 21]. In contrast, one of the contributions of our work is that we evaluated our IDS on a real traffic from a production SCADA network.

1.3 Contributions

In this paper we introduce a model-based IDS specifically built for Modbus/TCP. Our approach is based on a key observation: Modbus traffic to and from a specific PLC is highly periodic, with the same messages sent repeatedly in a fixed pattern. As a result, we can model each HMI-PLC channel by its own unique deterministic finite automaton (DFA).

Our IDS looks deep into the Modbus packets. The model captures detailed packet characteristics (not only the function codes but also the specific registers and coils that each message reads from or writes to). Based on the packet characterization the model captures the precise periodic traffic pattern between the HMI and PLC. Thus, our method 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. We designed an algorithm to automatically construct the channel's DFA based on about 100 captured messages.

A significant contribution is that we tested our approach on a production Modbus system controlling campus-wide power supply. We used our method to analyze over 120 hours of live traffic collected in two sessions several months apart.

Despite its high sensitivity, the system enjoyed a super-low false-positive rate: on 5 out of the 7 PLCs we observed a perfect match of the model to the traffic, without a single false alarm for 111 hours. Further, our system successfully flagged real anomalies that were caused by technicians troubleshooting the HMI system—and the system also helped uncover one incorrectly configured PLC.

2 Modbus over TCP/IP

2.1 Overview

Modbus has become a *de facto* standard for industrial control systems. Many Modbus systems implement the communication layer using TCP, as described in the Modbus over TCP/IP specification [4]. The specification defines an embedding of Modbus packets into TCP segments and assigns TCP port number 502 for the Modbus protocol. To maintain compatibility with Modbus over serial lines, the payload is limited to at most 253 bytes. Figure 2.1 illustrates the message structure of the Modbus protocol.

Transaction Identifier	Protocol Identifier	Length Field	Unit Identifier	PDU (data)	Checksum

Figure 2.1: Modbus/TCP Frame Structure

The Modbus protocol provides a simple master-slave communication mode between the devices. The *master* initiates the transactions (called queries) and the *slaves* respond by supplying the requested data to the *master*, or by taking the action requested in the query. Only one device can be declared *master*, usually the human machine interface (HMI), while the rest of the devices are *slaves* (usually Programmable Logic Controllers (PLCs) controlling devices like I/O transducers, valves, network drives, etc.). A response message is sent by the *slave* to all queries addressed to it individually. In heterogeneous networks, which comprise of both Modbus/TCP devices and serial Modbus devices, a gateway or a bridge can be 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 over seven bytes (see Figure 2.1), two of which are relevant to our work:

- Transaction Identifier - is a 2 bytes integer used for pairing the request and the response corresponding to a transaction. A unique Transaction ID is created on the request message from a *master*, which the *slave* includes in its response.
- Unit Identifier - is a single byte integer used to identify the Modbus slave associated with the transaction (relevant for the case of a Modbus gateway chaining several slaves).

2.2 The Modbus PDU

Each PLC provides an interface based on the Modbus data model. The data model is composed of "coil" (single bit) and "register" (16-bit) tables each containing elements numbered $[1..n]$. For each table the data model allows up to 65536 data items. The read or write operations of these items can access multiple multiple consecutive data items. The Modbus PDU has two fields that refer to the data model:

- Function Code - is a single byte integer in the range 1-127. The Modbus standard defines the meaning of 19 out of the 127 possible function codes . In our data sets, we have witnessed the appearance of only four different Function Codes, three *read* function codes (1,2,3) and one *write* (5).
- Payload - this field has variable size, limited to 252 bytes, and contains parameters that are specific to the function code. A *Read* request (Function Codes 1,2,3,4) payload consists of two fields: Reference Number and Bit/Word Count. The Reference Number field specifies the memory address to start

reading from. The Bit/Word Count field specifies the quantity of "memory object" units to be read. The corresponding response's payload consists of two slightly different fields: Byte Count and Data. The Byte Count field specifies the quantity of complete bytes of data. The Data field contains the values of the "memory objects" that were read. In addition to memory references, *Write* messages' payload includes fields specifying the values to be written.

A successful request execution is indicated by sending back a response packet, echoing the command's Function Code, followed by the relevant data (e.g., the read byte sequence in response to a read command). Failure is indicated by an exception response, which is a two-byte error response comprising of the original Function Code from the request PDU with its most significant bit set to logic 1.

2.3 Some Security Properties of Modbus/TCP

Note that the Modbus protocol does not defend itself in any way from a rogue master sending commands to the slaves. Further, Modbus does not have long-term session semantics: the whole protocol is a just single 2-message query-response. However, in all the examples we saw, the Modbus connection between the master and a specific slave is embedded into a single long-lived TCP connection. Moreover, at least one PLC we tested (Unitronics Vision350 [11]) can only accept a single TCP connection at a time on port 502. Thus, an attacker attempting to control an already controlled PLC would need to either hijack the existing TCP connection [12] and inject spoofed packets into the stream, or cause a Reset to the existing connection and start a new connection. PLCs that allow multiple concurrent connections on port 502 would be susceptible to much simpler attacks.

3 Accurate Protocol Modeling for Intrusion Detection

3.1 Overview

Our starting hypothesis is that a domain-specific Modbus IDS can be much simpler than a general-purpose IDS, and with much better false-positive rates. The intuition is that in a Modbus system entities are rarely added or removed: due to the network nature and purpose, we expect the emergence of new devices (neither PLCs nor HMI) to be very infrequent. Further, the HMI-to-PLC communication is extremely regimented device-to-device communication, with minimal human-initiated actions. A key ingredient is that the communication is highly **periodic**; The HMI repeatedly polls every PLC, at a fixed frequency, with a repeating sequence of commands. Thus we assume that the traffic pattern allows simple models with extremely high predictive power, which in turn admit very low false-positive IDS to be constructed.

S_i	The i-th state in the DFA. E.g., S_1 is the first state in the DFA.
s_i	The input-symbol leading to S_i.
q_i	The i-th *Query* message in the sequence.
Q_i	The state reached after q_i. E.g., Q_1 is the state reached after the first *Query* message in the sequence.
r_i	The i-th *Response* message in the sequence.
R_i	The state reached after r_i.E.g., R_1 is the state reached after the first *Response* message in the sequence.

Table 1: Notation

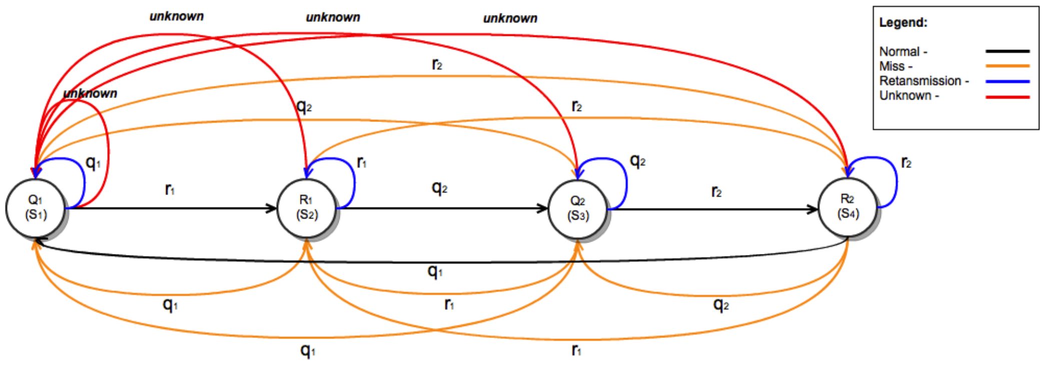


Figure 3.1: A DFA representing a 2-Query Modbus traffic pattern. The "normal" transitions follow the periodic traffic pattern consisting of two queries and their matching responses.

A preliminary inspection of our data sets yielded a few important observations that support the mentioned premises. As we shall see in Chapter 5, the system's static nature was validated by the near-fixed number of network entities. Throughout 120 hours of traffic recorded in our data set, over 5 months, we saw a single HMI and six PLCs (five of which were active during the whole period). Further, we recognized that the HMI communicates separately with each of the PLCs. Each connection is maintained as a long-term TCP connection, which is immediately restarted on any disconnection. This behavior enables us to handle every PLC individually.

3.2 Using Deterministic Finite Automata

Because of the clear patterns in the communication, we chose to model each HMI-PLC channel as a deterministic finite automaton (DFA). A classical DFA is a 5-tuple, $(Q, \sum, \delta, q_0, F)$, consisting of: a finite set of states (Q), a finite set of input symbols called the alphabet $(\sum)$, a transition function $(\delta : Q \times \sum \to Q)$, a 'start' state $(q_0 \in Q)$ and a set of 'accept' states $(F \subseteq Q)$. To apply a DFA approach to Modbus data we need to make the following adjustments:

1. We do not require the 'accept' states, since we want the IDS to continuously monitor an endless

repetitive stream. Instead, we opt for a Moore DFA, that associates an Action with every state transition in δ. Any deviation from the predicted pattern triggers a δ transition with an associated "error" action, which potentially causes an IDS alert (depending on the severity of the deviation). Further, we decided that the 'start' state will be defined as the state corresponding to the first *query* recognized in the periodic traffic pattern (this notation is further described in Table 1).

2. We need to select the Modbus features that identify a symbol in the alphabet $\sum$. At an extreme, we could use an overly naive alphabet of the two symbols $\sum=\{Query, Response\}$ and expect a pattern of $\{Query, Response\}*$. We chose to incorporate much more granularity into our model, by defining a symbol as a concatenation of several Modbus fields (see Section 3.4 below) totaling 33-bits. Despite the much longer alphabet, we did not suffer from state space explosion.

3.3 Channel Separation and Identification

The communication pattern to each PLC depends only on the HMI and is independent of the other PLCs behavior. Therefore, we decided to split the recorded traffic into separate *channels*, each containing only a single PLC's traffic. This enables us to model and analyze each PLCs behavior separately without artificially increasing the model's state space. This channel separation is easy to do based on the PLC's IP address.

A *channel* is defined by the tuple $(Master\,IP, Slave\,IP)$ and is identified after the recognition of a Modbus packet (port 502 by default). If the master IP is different from the (single) expected IP an alert "UNEXPECTED MASTER" will be raised. Similarly, if the slave IP is not one of the expected slaves IPs, an alert will be raised. These conditions are equivalent to the Digital Bond [2] Snort rules 1111006, 1111007.

As mentioned in Section 2, in some SCADA networks a Modbus gateway is used to chain several PLCs. In our network, we observed PLC #5 functioning as a gateway chaining two PLC. The communication between each of the chained PLCs and the HMI is independent (similar to the communication between a the HMI and non-chained PLCs). Recall that the Unit Identifier fields is used to address chained PLCs, thus we execute a finer channel definition and separation using the 3-tuple $(Master\,IP, Slave\,IP, Unit\,Identifier)$. This definition enables us to treat each chained PLC individually, which in our case separates between PLC #5.1 and PLC #5.2. Reference to a new Unit Identifier in a query message will raise an alert. Note that the Digital Bond Modbus Snort rules do not catch such an anomaly.

3.4 States and Input Symbols

Our basic observation is that the HMI-PLC traffic pattern for a given PLC is periodic, repeating the same sequence of queries (and matching responses) over and over. E.g., Our data shows that for PLC #2 the HMI sends a sequence of 3 fixed queries (and receives their matching responses) every 30ms, and this pattern of 6 messages is maintained for many hours. Once we identify the pattern's length (6 in the above example), we can define a DFA similar to the DFA depicted in Figure 3.1. Hence, 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*. Respectively, states that are reached after response messages are called *R-states*.

Recall that a Modbus query consists of the following fields: Transaction Identifier (T.ID), Function Code (FC), Reference Number (RN), and bit/word[1] count (Count).

We define a symbol in the alphabet Σ as a 4-tuple containing the above mentioned fields except the T.ID . This leads us to 33-bit symbols (1-bit for Q/R, 8-bit for function code, 16-bit reference number and 8-bit for bit/word count). Responses do not include the reference number so those 16 bits in the symbol are always 0.

Input symbols are categorized into two groups: *Known* and *Unknown*. The *Known* group consists of all the input symbols that were observed during the learning phase (described in Chapter 4), and have a matching DFA state. Respectively, the *Unknown* group consists of the input symbols that do not have a matching DFA state or were not observed during the learning phase.

3.5 The Transition Function

The 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 our premises about the network traffic characteristics by matching the correct state and operation to the given base-state and input-symbol. We define four types of transitions. We denote current position as S_i and the received input-symbol as s_j :

- **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, we save the message's T.ID. If the symbol is a response, we compare the current message's T.ID with the saved T.ID. If the IDs do not match, we increment a "T.ID mismatch" counter. In our data we observed only a handful of T.ID mismatches (less than 0.004% of the packets), and these were all caused by dropped packets in the capture mechanism.

  As part of the Normal transition we implement the in-packet validation tests suggested by [2] and

[1]Depending on the *Function Code* field value. Some Function codes are followed by *bit count* while other are followed by *word count*.

[15], most importantly verifying that the packet payload length is indeed at most 252 bytes. This mechanism flags buffer-overflow attempts against the HMI [via too-long fake responses from PLCs] or against the PLCs [via too-long queries from the HMI]. Note that we do not need to explicitly test the actual packet length against the in-packet Count value, or for mismatches between the requested Count and the supplied length in the response: since the Count field is always part of the symbol, any attempt to send too much (or too little) data would cause the packet to trigger an Unknown transition (see 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 this occurrence we add a self-loop to the DFA,

  $\mathit{Dest\ State} = \mathit{Base\ State} = S_i$

  Retransmissions occur normally in TCP traffic due to momentary congestion, and they do not indicate a real anomaly in the Modbus operation. Thus, the action we take is just to increment a counter. Note that if the pattern includes two identical symbols, we will get a state with 2 different transitions for the same symbol (a "normal" transition forward, and a "retransmission" self loop). We resolve such non-determinism in run-time by preferring the "normal" transition over the self-loop "retransmission" transition.

- **Miss** - A "miss" is an occurrence of a known symbol s_j which appears at 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 will skip sending some packet in the normal pattern, and even more unlikely that the PLC will ignore a query. Therefore we chose to handle a Miss event by a transition to the closest forward state (modulo *Pattern_Length*) that follows the normal s_j symbol. Again, since in our view a Miss is a relatively benign anomaly, and most probably an artificial anomaly introduced by the IDS packets capture, we only increment a "Miss" counter.

- **Unknown** - The most serious anomaly is an "unknown" symbol appearing. At worst, an unknown symbol can be an indication of a malicious packet injected into the TCP stream. However, other interpretations are also possible a-priori: An unknown query could indicate a human operator's action, or an (un-modeled) automatic response by the HMI to some condition observed in previous data, or an indication that the modeled pattern is too short to capture infrequently sent queries. An unknown response could indicate a faulty PLC which is responding with the wrong Function Code, or sending back the wrong amount of data. Whatever the interpretation of the unknown symbol may be, technically we transition back to the first state (in the hope that the pattern will resynchronize), increment the "unknown" counter, and raise an alarm with the value of the symbol. Naturally, an unknown "write" Function Code is more severe than an unknown "read".

4 Creating the Model

4.1 Automatic model generation

One of the novel aspects of our approach is that we can automatically learn the model without any labeling of the training data. All we require is a clean capture of normal traffic that is longer in length than the pattern. We start the learning phase by capturing a fixed number of packets, indicated by *Learning_ Window_ Size*. We take an inventory of the identified queries and responses in the window, and perform several checks on that inventory. Those checks include verifying that each query has a valid response and verifying that each response has a preceding query. Then, we use an iterative method to create the smallest DFA that models the sniffed Modbus packets (See Algorithm 1).

We start with an initial estimate of 2 for the pattern length (i.e., one query and one response, the shortest possible legitimate pattern). In each iteration, we define the current pattern candidate as the first *Pattern_ Length* Modbus messages, starting with a query message, in the window. From this candidate pattern, we construct a DFA as described in Sections 3.4 and 3.5. Then, we run the created DFA against *Validation_ Window_ Size* captured Modbus messages, and count "misses", "retransmissions", and "unknowns" as in Section 3.5 (this window is assumed to be clean of unexpected network events or activities, having no anomalies). From these counters we define a performance value as:
$P = \frac{normals}{total} = \frac{normals}{normals+misses+retransmissions+unknowns}$.

If P is below a set threshold we conclude that *Pattern_ Length* was too small, thus we increment it by 2 and start a new iteration. If *Pattern_ Length* exceeds *Learning_ Window_ Size*, we stop with a failure.

4.2 Setting the Threshold

Each channel, hence each PLC, is characterized by its own periodic pattern length. Let us denote the periodic pattern length as k and the candidate pattern length as n. For each channel, we need to discover k separately. Our performance threshold, mentioned in Algorithm 1, should be defined such that it differentiates between the correct pattern length from other shorter/larger 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. Using the following observations, we were able to analytically define a threshold that accomplishes the desired differentiation. We make the following assumptions: (a) We have a clean validation window of length V. For simplicity, we assume that the validation window size obeys $V \bmod k = 0$ and that $V \to \infty$; and (b) The

Algorithm 1 Pattern Modeling

```
1. $Pattern\_Length \leftarrow 2$
2. DFA $\leftarrow$ DataLearning(Pattern_Length)
3. performance_value $\leftarrow$ ModelValidation(DFA)
4. while (performance_value $>$ Threshold) and
   (Pattern_Length $<$ Learning_ Window_ Size):
    (a) Pattern_Length $\leftarrow$ Pattern_Length $+2$
    (b) DFA $\leftarrow$ DataLearning(Pattern_Length)
    (c) performance_value $\leftarrow$ ModelValidation(DFA)
5. if (Pattern_Length $\geqq$ Learning_ Window_ Size): "FAILED"
6. else: return DFA
```

expected symbol	a	b	c	d	a	a	c	d	a	a	c	d	a	a	c	d
input symbol	a	b	c	d	a	b	c	d	a	b	c	d	a	b	c	d
transition type	N	N	N	N	N	$\underline{\text{M}}$	N	N	N	$\underline{\text{M}}$	N	N	N	$\underline{\text{M}}$	N	N

Table 2: Model performance for an input (a,b,c,d)* with k=4, and a candidate pattern (a,b,c,d,a)* with n=5 and i=1. Notation: N - normal ; M - miss ; U - unknown. Ignoring the first 4 symbols we can see that $P = \frac{i \cdot k - 1}{i \cdot k} = \frac{3}{4} = 75\%$.

pattern consists of k distinct messages.

Given a periodic sequence of distinct messages of a length k and a clean validation window of a size $Validation\_ Window\_ Size = V$, we construct a DFA in the size of the candidate length n. The DFA is constructed by the method mention in chapter 3. For each candidate, we evaluate the DFA's performance on the validation window. Consider the following cases:

- The candidate length is shorter than the actual pattern length, i.e., $n \leq k$. Then the model will mistakenly recognize the last $k - n$ message in the periodic sequence as "unknowns". Thus, for each appearance of the periodic pattern our model will count n 'normal' transitions and $k - n$ "unknown" transitions (corresponding to the unknown messages), resulting with $P = \frac{n}{k}$.

- The candidate length is a multiple of the actual pattern length, i.e., $n = i \cdot k$ where $i \in N$. The the model will contain multiple repetitions of the complete periodic sequence. Thus, no "unknowns" will occur since the DFA contains all the messages appearing in the validation window. Further, since the DFA contains an exact multiple of the period sequence no 'misses' or 'retransmissions' will occur either. Consequently, $P = 1$.

- The candidate length is longer than the actual pattern length but is not a multiple of the the actual pattern length. Thus, $n = i \cdot k + r$ for $1 \leq r < k$. Then the first $i \cdot k + r$ symbols will trigger "normal" transitions. Then the DFA will expect symbol s_1 but will encounter symbol s_{r+1} , causing a "miss". However, recall that a miss transition has a next state which is the closest forward state matching the input, i.e., the DFA will transition to state S_{r+1} - effectively resynchronizing the expected pattern with the input. All subsequent symbols will again trigger "normal" transitions, until s_{2ik+r} triggers another "miss", and so forth. In every block of $i \cdot k$ input symbols (except the first) the DFA will trigger a single "miss" and $i \cdot k - 1$ "normal" transitions. Thus when $V \to \infty$ we obtain $P = \frac{i \cdot k - 1}{i \cdot k}$ - see Table 2. Note that the performance value is independent of r, we obtain the same $P = \frac{i \cdot k - 1}{i \cdot k}$ for all values $n = i \cdot k + 1, ..., i \cdot k + (k - 1)$.

In summary, we see that when the pattern comprises of k distinct symbols, the input is perfectly clean and the validation window $V \to \infty$, the performance value as a function of the candidate length n obeys:

$$P = \begin{cases} \frac{k}{n} & n \le k \\ 1 & n = i \cdot k, \quad i \ge 1 \\ \frac{i \cdot k - 1}{i \cdot k} & i \cdot k + 1 \le n \le i \cdot k + (k-1), \quad i \ge 1 \end{cases} \tag{4.1}$$

We would like to tune the threshold T so that it causes Algorithm 1 to stop for $n{=}k$. To achieve this we need to set $T = \frac{k-1}{k}$ - except k is unknown to us. However, as long as $k \le n$ the sequence $P(n)$ is increasing, so it suffices to set T high enough to <u>not</u> accept $n = k - 1$: i.e., setting $T = \frac{n}{n+1}$ and having Algorithm 1 stop when $P > T$ is enough. Note that the threshold provides less discrimination as n grows, since the gap between $\frac{n}{n+1}$ and 1 shrinks.

5 Data Acquisition

5.1 Overview

Due to their sensitivity, real data sets of SCADA networks are usually kept confidential and it is quite difficult to get hold of real data sets. Therefore, many researchers rely on data sets extracted from SCADA testbeds in their work. In fact, we are unaware of any publicly accessible traces of Modbus/TCP that consist of more than a handful of packets.

One of contributions 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 uses a Modbus/TCP-based system to monitor the campus power grid, and that this system's communication uses the campus-wide IP network. With the assistance of the university's CISO we were allowed to tap into the Modbus communication and record it at two points in time, producing two data sets. Basic information about the data we collected appears below.

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

5.2 Preliminary Analysis

One of our research goals was to keep our method's network-dependency as low as possible by not using any prior knowledge about it. Therefore, a preliminary network analysis was needed to be performed in order to produce basic insights. The pre-analysis focused mainly on gathering SCADA entities identification and traffic statistics. The preliminary analysis was performed using WireShark [8] Analysis and Statistics built-in features and using automated scripts written in Python using Impacket [3] and Pcapy [6] modules for network packets handling. After the pre-analysis was done, we met the facility manager to validate our findings and got the vendor and model names of the system components.

In data set #1 we observed 4 Modicon [5] PLCs and a single Satec [7] PLC (with two chained unit-ids), all controlled by an Afcon [1] Pulse HMI. In data set #2 we observed the same PLCs with the appearance of one additional Modicon PLC.

Using a splitting procedure written in Python, we split the primary data files into sub-files, each containing the packets of a certain time-frame. Data set #1 was split into 630 time-frames, 10 MB each (equivalent to 2 minutes). Data set #2 was split into 1340 time-frames, 26 MB each (equivalent to 5 minutes). During our analysis, we often used these time-frames as a basic unit for calculations and comparison.

6 Model Validation

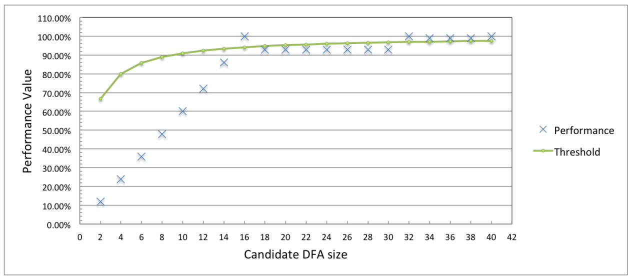
To validate our DFA-based model we implemented our DFA construction method in standard Python using Impacket [3] and Pcapy [6] modules for network packet handling. The analysis results as well as anomalous-suspected traffic were verified and validated against the network activity-log with the facility manager.

6.1 Model Creation with Automatic Threshold Tuning

Running our Algorithm 1 with an auto-tuned threshold (recall section 4.2) on our data sets with the parameters values as in Figure 6.1(a) yielded very good results accurately identifying periodic pattern length for each one of the PLCs for both data sets. Figure 6.1(b) shows the method performance for PLC #1. We see that on data set #1 the method successfully identified the periodic pattern length $k = 16$. Note the local maxima at n=16, 32.

Parameter	Value
Learning_ Window_ Size	50 packets
Validation_ Window_ Size	100 packets
Performance Threshold	$\frac{n}{n+1}$

(a)



(b)

Figure 6.1: Parameter values in model creation procedure, and Performance value vs. candidate DFA size for PLC #1 on data set #1 .

6.2 Basic Model Validation

Basic model validation deals with the model's ability to represent the 'normal' network traffic using a DFA structure. We use two parameters to measure the DFA quality: *Pattern_ Length*, and "unknown", "miss" and "retransmission" rates. Recalled that *Pattern_ Length* is the smallest integer whose performance passes the threshold. Successfully fitting a *Pattern_ Length* that makes the DFA pass the performance threshold, demonstrates that the DFA represents the traffic captured in the *'validation window'* accurately. *Pattern_ Length* = *Validation_ Window_ Size*, will obviously pass the threshold successfully.

Table 3 summarizes the *Pattern_ Length* results for the PLCs. We see that for each of the PLCs our method successfully constructed DFA representing very short periodic patterns. E.g., on data set #1, the largest DFA was observed on PLC #1, with a *Pattern_ Length* of 16. Hence, the network traffic between PLC #1 and the HMI follows a periodic pattern comprising of 8 queries and their matching responses.

The second set of parameters measuring the model's quality are low "unknown" "miss" and "retransmission" rates. Figure 6.2 (left column) shows that except for few distinct peaks throughout the entire model run, all three anomaly counters were extremely low and represent only verified network congestion and packet drops. In fact, 625 of the 630 time frames (of data set #1) are "quiet": the "unknown" counter was exactly 0, and both the "miss" and "retransmission" counters was below 15 per time-frame. This validation was performed on each one of the PLCs, with similar results .

6.3 Anomaly Detection

In data set #1 the "unknown" rate was very low for all PLCs– at most 0.39% of the packets. However, we can see that the "unknown" symbols are not evenly distributed over time: in fact 97.7% of the time-frames in data set #1 are completely quiet. Figure 6.2 (left column) clearly shows that in data set #1 there were 2 interesting periods of anomalous activity: near time-frames #84 and #460. Furthermore, Figure 6.2 (right

Data set #1

PLC	*Pattern_ Length*
#1	16
#2	6
#3	6
#4	6
#5.1	2
#5.2	2
#6	-

Data set #2

PLC	*Pattern_ Length*
#1	18
#2	4
#3	6
#4	6
#5.1	2
#5.2	2
#6	6

Table 3: *Pattern_ Length* results for the PLCs recognized over Data Set #1 and Data Set #2.

column) shows that these events affected all the PLCs at the same time, making them even more suspicious. As we discovered later, these were not false-positives but actual anomalies that our system flagged (see Section 6.4).

Note that correlating anomalous activity observed in multiple devices is a well known IDS mechanism. An IDS following our approach is a suitable feed for event-correlation systems such as EMERALD [10] or a commercial system such as ArcSight [9].

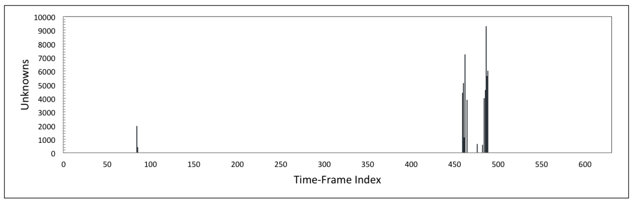
Recall that our transition function is defined such that after each "unknown" input-symbol our model's state is changed to the *start-state*. Due to the arbitrary "unknown" symbol position (in the periodic sequence) the next transition will likely be either a "miss" or a "retransmission". Therefore, the three counters are technically correlated due to the way we construct the model. Figure 6.2 (left column) clearly shows this correlation, with obvious spikes in all three counters near time frames #84 and #460.

6.4 Real Anomalies in Data Set #1

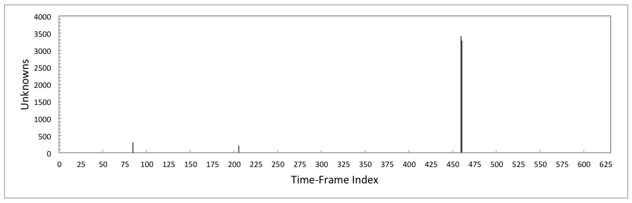
Once we finished analyzing the network using our modeling method, we met with the facility manager and examined our suspicious-labeled messages and events versus the network logs. We discovered that all the "unknown" transition were verified to be indeed suspicious (and not a misdetected false alarm). The prominent interrupts depicted Figure 6.2 were verified and were found to be indication of a technician, who was troubleshooting problems with the system that day.

7 Details of Data Set #2

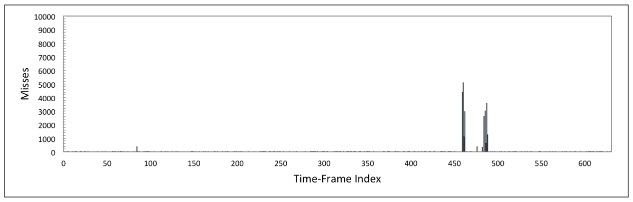
Data set #2 was recorded five months after the recording of data set #1. During these months, the SCADA system was successfully upgraded and stabilized by technicians. The changes made, as part of the upgrade process, caused several significant effects on the SCADA network traffic. First, in addition to the 5 PLCs we saw in data set #1, a new PLC appeared. The new PLC is another Modicon PLC, similar to PLCs #2-4,



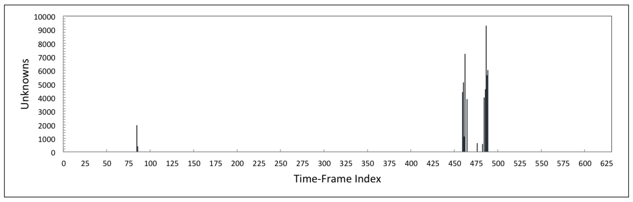
Unknowns



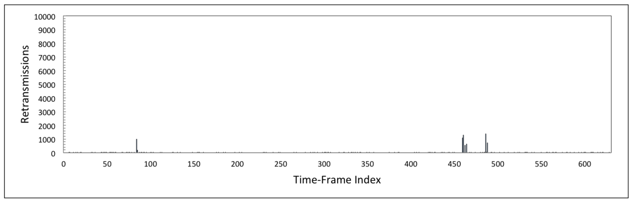
PLC #1



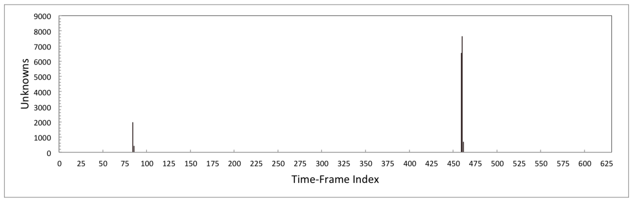
Misses



PLC #2



Retransmissions



PLC #3

Figure 6.2: Left column: correlation between event types in PLC #2 on data set #1. Right column: correlation between Unknowns across different PLCs.

and is labeled as PLC #6. As seen in Table 3, we also observed some minor changes in the traffic pattern of two PLCs: The *Pattern_Length* of PLC #1 grew to 18, adding an additional query (and its matching response) to the periodic sequence, and PLC #2 dropped a query (and its matching response) of the periodic sequence, bringing it to a shorter $Pattern\_Length = 4$.

When we ran our method on data set #2, we reached distinctly different performance statistics . We identified two opposite effects in the performance in comparison to data set #1. The first effect shows a perfect modeling of the network traffic for 5 of the 7 PLCs. For each of these PLCs, our method modeled the network traffic perfectly for 111 hours, without any "unknown" messages. Thus for these PLCs we can conclude that despite the high sensitivity of the DFA approach, our system did not raise *any* false alarms.

The second effect, concerning only PLC #1, shows a significant increase in the "misses" and "unknowns" frequency: from 0.09% in data set #1 to 0.9% in data set #2. Even more problematic is that the fraction of quiet time-frames dropped to only 66% . In other words, the "unknown" events are not localized to a few anomalous time frames as in data set #1, but are spread throughout the entire data set. A closer examination of "unknowns" events versus time, reveals that the control of PLC#1 is operating with three separate time periods. Besides the high-frequency pattern that is well modeled by the DFA, we observed two other periodic patterns that are much slower: a low-frequency periodic pattern with a period $T_1 = 24 hours$, observed 4 times in data set #2, and a mid-frequency periodic pattern with a period $T_2 = 15min$, observed 446 times in data set #2. Because we used a 5-minute time frame for data set #2, the quarter-hourly pattern caused the effect that only 66% of time frames were quiet: it occurred in one out of every three time frames. Our facility manager verified that both patterns are "normal", noting that the daily periodic pattern ($T_1 = 24hours$) is used to reset various PLC counters, and the quarter-hourly pattern ($T_2 = 15min$) is for averaging a set of control process counters.

We have extended our approach to deal with multi-period patterns. Using a multi-level DFA we have been able to reduce the number of "unknowns" in PLC#1 down to only 0.0045%. Details omitted from this extended abstract due to space constraints.

8 Concluding Remarks

We believe that DFA-based approach has been successful in two ways. On one hand, despite its high sensitivity, the system enjoyed a super-low false-positive rate. On the other hand, the system successfully flagged real anomalies—that do not raise any alerts from the Snort rules of [2]. Thus we are encouraged by the system's performance so far and we believe the approach has merit.

Evaluating an IDS on live traffic from a production system provides valuable insights, but has some inherent limitations, which we plan to address in future work:

- We did not attempt to inject any malicious traffic into the network to avoid interfering with the SCADA system's operation. We would like to experiment further with our approach in a lab environment where we can test more aggressive scenarios.
- We only evaluated our approach on a single Modbus/TCP system. To further validate our observations we would like to test our approach in other Modbus/TCP systems.

References

[1] Afcon software and electronics ltd. `http://www.afcon-inc.com/`.

[2] Digitalbond. `http://www.digitalbond.com`.

[3] Impacket - network protocols constructors and dissectors. `http://oss.coresecurity.com/projects/impacket.html`.

[4] *Modbus Messaging Implemetion Guide V1.0b.* `http://www.modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf`.

[5] Modicon. http://www.schneider-electric.com/.

[6] Pcapy - python pcap extension. http://oss.coresecurity.com/projects/pcapy.html.

[7] Satec. http://www.satec-global.com.

[8] Wireshark - network protocol analyzer. www.wireshark.org.

[9] HP ArcSight security intelligence. http://www.hpenterprisesecurity.com/products/hp-arcsight-security-intelligence.

[10] Event monitoring enabling responses to anomalous live disturbances (EMERALD). http://www.csl.sri.com/projects/emerald/.

[11] Unitronics vision350. http://www.unitronics.com/Series.aspx?page=Vision350.

[12] S. M. Bellovin. Security problems in the TCP/IP protocol suite. *SIGCOMM Comput. Commun. Rev.*, 19(2):32–48, April 1989. ISSN 0146-4833. doi: 10.1145/378444.378449. URL http://doi.acm.org/10.1145/378444.378449.

[13] L. Briesemeister, S. Cheung, U. Lindqvist, and A. Valdes. Detection, correlation, and visualization of attacks against critical infrastructure systems. In *Eighth Annual International Conference on Privacy Security and Trust (PST)*, pages 17–19, 2010.

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

[15] S. Cheung, B. Dutertre, M. Fong, U. Lindqvist, K. Skinner, and A. Valdes. Using model-based intrusion detection for SCADA networks. In *Proceedings of the SCADA Security Scientific Symposium*, pages 127–134, 2007.

[16] W. Chunlei, F. Lan, and D. Yiqi. A simulation environment for SCADA security analysis and assessment. In *International Conference on Measuring Technology and Mechatronics Automation (ICMTMA)*, volume 1, pages 342–347. IEEE, 2010.

[17] I.N. Fovino, A. Carcano, T. De Lacheze Murel, A. Trombetta, and M. Masera. Modbus/DNP3 state-based intrusion detection system. In *24th IEEE International Conference on Advanced Information Networking and Applications (AINA)*, pages 729–736. Ieee, 2010.

[18] W. Gao, T. Morris, B. Reaves, and D. Richey. On SCADA control system command and response injection and intrusion detection. In *eCrime Researchers Summit*, pages 1–9. IEEE, 2011.

[19] B. Genge, C. Siaterlis, I. Nai Fovino, and M. Masera. A cyber-physical experimentation environment for the security analysis of networked industrial control systems. *Computers & Electrical Engineering*, 2012.

[20] A. Giani, G. Karsai, T. Roosta, A. Shah, B. Sinopoli, and J. Wiley. A testbed for secure and robust SCADA systems. *ACM SIGBED Review*, 5(2):1–4, 2008.

[21] A. Hahn, B. Kregel, M. Govindarasu, J. Fitzpatrick, R. Adnan, S. Sridhar, and M. Higdon. Development of the PowerCyber SCADA security testbed. In *Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research*, page 21. ACM, 2010.

[22] D.J. Kang and H.M. Kim. Development of test-bed and security devices for SCADA communication in electric power system. In *31st International Telecommunications Energy Conference (INTELEC)*, pages 1–5. IEEE, 2009.

[23] M. Mallouhi, Y. Al-Nashif, D. Cox, T. Chadaga, and S. Hariri. A testbed for analyzing security of SCADA control systems (tasscs). In *IEEE Innovative Smart Grid Technologies (ISGT)*, pages 1–7. IEEE, 2011.

[24] E. Naess, D.A. Frincke, A.D. McKinnon, and D.E. Bakken. Configurable middleware-level intrusion detection for embedded systems. In *25th IEEE International Conference on Distributed Computing Systems*, pages 144–151. IEEE, 2005.

[25] Martin Roesch. Snort - lightweight intrusion detection for networks. In *Proceedings of LISA '99: 13th Systems Administration Conference, USENIX*, 1999.

[26] C.H. Tsang and S. Kwong. Multi-agent intrusion detection system in industrial network using ant colony clustering approach and unsupervised feature extraction. In *IEEE International Conference on Industrial Technology (ICIT)*, pages 51–56. IEEE, 2005.

[27] A. Valdes and S. Cheung. Communication pattern anomaly detection in process control systems. In *IEEE Conference on Technologies for Homeland Security (HST)*, pages 22–29. IEEE, 2009.

[28] A. Valdes and S. Cheung. Intrusion monitoring in process control systems. In *42nd Hawaii International Conference on System Sciences (HICSS)*, pages 1–7. IEEE, 2009.

[29] D. Yang, A. Usynin, and J.W. Hines. Anomaly-based intrusion detection for SCADA systems. In *5th Intl. Topical Meeting on Nuclear Plant Instrumentation, Control and Human Machine Interface Technologies (NPIC&HMIT 05)*, pages 12–16, 2006.