Chapter 4

MODELING MESSAGE SEQUENCES FOR INTRUSION DETECTION IN INDUSTRIAL CONTROL SYSTEMS

Marco Caselli, Emmanuele Zambon, Jonathan Petit and Frank Kargl

Abstract Compared with standard information technology systems, industrial control systems show more consistent and regular communications patterns. This characteristic contributes to the stability of controlled processes in critical infrastructures such as power plants, electric grids and water treatment facilities. However, Stuxnet has demonstrated that skilled attackers can strike critical infrastructures by leveraging knowledge about these processes. Sequence attacks subvert infrastructure operations by sending misplaced industrial control system messages. This chapter discusses four main sequence attack scenarios against industrial control systems. Real Modbus, Manufacturing Message Specification and IEC 60870-5-104 traffic samples were used to test sequencing and modeling techniques for describing industrial control system communications. The models were then evaluated to verify the feasibility of identifying sequence attacks. The results create the foundation for developing "sequence-aware" intrusion detection systems.

Keywords: Industrial control systems, sequence attacks, intrusion detection

1. Introduction

Critical infrastructure assets such as power plants, electric grids and water treatment facilities have used control systems for many decades; however, until the turn of the century, they were primarily standalone systems. The Internet and network convergence have brought about many changes to critical infrastructure assets, the most important being their transformation from standalone systems to highly interconnected systems. This transformation has introduced advantages and disadvantages. On one hand, it facilitates the remote monitoring and management of industrial processes. On the other hand, traditional information technology attacks can be launched from afar, includ-

M. Rice, S. Shenoi (Eds.): Critical Infrastructure Protection IX, IFIP AICT 466, pp. 49–71, 2015.
DOI: 10.1007/978-3-319-26567-4_4

ing over the Internet, to compromise industrial control systems and the critical infrastructure assets they manage.

This is the case of denial-of-service and distributed denial-of-service attacks. These attacks can target a specific device in an industrial control network and flood it with a massive number of packets until it is no longer able to operate normally. This can reduce or eliminate operator situational awareness and eventually impact the coordination and control of infrastructure assets, potentially affecting the larger infrastructure and connected infrastructures, leading to serious consequences to industry, government and society.

Another example involves semantic attacks. Unlike standard cyber attacks, semantic attacks exploit knowledge of specific control systems and physical processes to maximize damage. Stuxnet [4, 16] is probably the most well-known attack of this type. Meanwhile, numerous reports from the U.S. ICS-CERT have described exploits on industrial devices, such as programmable logic controllers and SCADA servers, that are triggered by carefully-crafted messages (see, e.g., [9]). Sequence attacks are a type of semantic attack. Instead of using modified message headers or payloads, these attacks employ misplaced messages in industrial control system communications to cause targeted devices to malfunction or even strike directly at physical processes.

Detecting sequence attacks relies on two assumptions: (i) industrial control system communications can be monitored; and (ii) industrial control system communications are generally regular over time. Traditional industrial technology networks rarely have regular network traffic due to their high variability (e.g., web users downloading different kinds of content from the Internet or interacting among themselves). However, even if an industrial control network maintains the same topology (i.e., a device always communicates with the same devices) [6], no checks are performed – nor is evidence is maintained – that the same sequences of messages are present in the inter-device communications.

Another problem is that traditional network intrusion detection systems generally search for unusual messages and rarely focus on message sequences, causing sequence attacks to go unnoticed. Specification-based intrusion detection systems can deal with sequence attacks based on the misuse of communications protocols. However, they require a precise and comprehensive documentation of system operation (e.g., the TCP/IP protocol suite [22]).

This chapter describes several sequence attack scenarios against industrial control systems. Real Modbus, Manufacturing Message Specification (MMS) and IEC 60870-5-104 traffic samples are used to evaluate the feasibility of sequence analysis and detection. Indeed, the goal is to identify all the traffic data needed to model and analyze the behavior of industrial control protocols over time. Furthermore, the feasibility of "sequence-aware" detection is investigated by identifying model-related information that can be leveraged to detect malicious activities. The implementation of sequence-aware intrusion detection is left as future work because the objective of this research has been to identify plausible sequence detection methodologies that can be used to differentiate attack traffic from normal traffic in industrial control systems.

2. Background

Industrial control systems include supervisory control and data acquisition (SCADA) systems, distributed control systems (DCSs) and generic control systems such as skid-mounted programmable logic controllers (PLCs) [23]. Industrial control networks usually interconnect a field network and a process network. The field network hosts devices that are directly connected to the physical process. The process network hosts the servers that supervise the control of the physical process. Field devices include sensors, actuators, remote terminal units and programmable logic controllers. Process networks include SCADA servers, human-machine interfaces (HMIs) and engineering workstations.

Industrial control systems use special communications protocols on top of TCP/IP, open protocols like Modbus, MMS and IEC 60870-5-104 and proprietary protocols such as Siemens S7. This research focuses on Modbus, MMS and IEC 60870-5-104 because, in addition to being widely used, these protocols demonstrate different communications patterns and behaviors (e.g., synchronous vs. asynchronous communications, and pushing vs. polling paradigms).

- **Modbus:** This application layer protocol uses a polling client/server scheme [19, 24]. SCADA servers (i.e., masters) always act as clients and initiate communications by sending requests to programming logic controllers (i.e., slaves) and wait for responses. Common Modbus commands can, for example, instruct a server to read or write values in its memory-mapped registers.

- **Manufacturing Message Specification (MMS):** This protocol implements the seven layers of the ISO/OSI stack, even when it operates on top of TCP/IP [12, 13]. MMS is a client/server protocol with synchronous or asynchronous communications patterns. The protocol defines read and write control operations for a set of standard objects, a set of messages to be exchanged and a set of encoding rules that map the messages.

- **IEC 60870-5-104 (IEC104):** This application layer protocol operates on top of TCP/IP [10, 11]. The protocol mostly uses asynchronous balanced or unbalanced data transfer modes. In the balanced mode, a master or slave can initiate communications while the unbalanced mode only allows a master to initiate communications, In the remainder of this chapter, the IEC 60870-5-104 protocol is referred to as IEC104.

3. Sequence Attacks

This section presents four sequence attacks that are classified according to the targeted industrial control system component (i.e., device or physical process) and the type of compromise (i.e., manipulation of the timing or order of messages).

With regard to the first dimension (i.e., attack targets), this section distinguishes between attacks targeting the implementations of protocol stacks in

devices and attacks targeting the industrial processes controlled by the devices. Specifically, the first, and more traditional, class of attacks exploit vulnerabilities in protocol stack implementations. The second class of attacks, which are specific to industrial control systems, attempt to divert or take control of industrial processes by leveraging the lack of integrity and authentication mechanisms. To take control of a process, an attacker may either reprogram the logic executed by a programmable logic controller [18] (e.g., as in the case of Stuxnet), or directly control the process from the network using the same control messages used by legitimate operators. This research focuses on the second type of attacks that are less explored in the literature.

With regard to the second dimension, this research distinguishes between attacks that send messages or commands in incorrect (malicious) order and attacks that send messages or commands with incorrect (malicious) timing. The first type of attacks violate the state machines that underlie application protocols or send sequences of messages or commands that move processes to unsafe states. The second type of attacks leverage the limitations of embedded devices in processing input data or exploit weaknesses of process equipment (e.g., motors or valves) by changing their operational states in ways not foreseen by their manufacturers.

Based on these two dimensions, four attack scenarios are described and a simple, yet realistic, example is provided for each of the four scenarios:

- **Message-Order-Based Device Compromise:** The majority of application level protocols used to manage programmable logic controllers provide for loading and storing logic programs from engineering workstations with network connectivity to programmable logic controllers. The load and store functionalities are typically achieved by sending sequences of messages. The sequence of messages for uploading a logic program to a programmable logic controller typically involves: (i) locking the programmable logic controller; (ii) stopping the running program(s); (iii) deleting the existing program(s); (iv) transferring the program code to the programmable logic controller; (v) creating new program(s) with the new program code; (vi) starting the new program(s); and (vi) unlocking the programmable logic controller. Each step is achieved by sending one of more messages to the programmable logic controller. Experiments in an industrial control laboratory environment have revealed that it is possible to attack some programmable logic controllers merely by sending some of these messages in an inconsistent order. For example, sending a (valid) start program message when the program is still running causes an error that is not properly handled by the programmable logic controller firmware, which causes the programmable logic controller to crash.

- **Message-Order-Based Process Compromise:** Carcano et al. [20] describe an example attack scenario for a process system comprising a pipe with high pressure steam. The pressure is regulated by two valves (V1 and V2). An attacker with the appropriate access sends a write

message to the programmable logic controller to completely close valve V2 and another write message to completely open valve V1. This maximizes the flow of steam into the pipe and maximizes the pressure in the pipe because the incoming steam cannot exit the pipe. Both these commands are perfectly legal when considered individually, but they bring the system to a critical state because they are sent in the wrong sequence.

- **Message-Timing-Based Device Compromise:** Many embedded devices used in industrial control systems have limited computing capabilities and have weak protocol stack implementations (because they incorrectly assume that all devices use the protocols correctly). As a result, it is fairly easy to mount attacks that exhaust the resources of these embedded devices by simply flooding the network with (valid) protocol messages (e.g., TCP SYN messages). However, laboratory experiments conducted as part of this research have revealed that the same effects can be achieved using application-level messages. This is particularly true for programmable logic controllers that use UDP-based application protocols. Flooding these devices with application-level messages that require expensive operations (e.g., diagnostic functions) quickly exhausts programmable logic controller resources, compromising their ability to complete their scan-loops in real time and eventually leading to complete resets of the devices.

- **Message-Timing-Based Process Compromise:** A report by the U.S. President's Commission on Critical Infrastructure Protection [21] describes an example attack scenario involving a water distribution facility. In this scenario, major control valves on the water pipeline are rapidly opened and closed to cause water hammer, resulting in a number of simultaneous water main breaks. Such an attack could be carried out by rapidly sending a sequence of write messages that direct programmable logic controllers to open and close the valves.

4. Sequences and Sequence Events

Detecting sequence attacks requires the ability to extract a sequence of messages from network traffic and identify information that is needed to construct intrusion detection systems. This section investigates the process of transforming a set of network frames into ordered lists of network events that represent device communications.

The easiest way to define an event in the context of network communications is to consider all the traffic frames one by one. However, all traffic frames are not equally important and not all frames need to be included in a sequence. Moreover, all the attacks presented in Section 3 involve a single connection between a sender and a receiver (i.e., one TCP stream). Therefore, it is necessary to group traffic frames into communications channels. The following definitions are used in the ensuing discussion:

Definition 1: *A sequence* $\{m_{t_n}\}$ *with* $n \in [0, \infty[$ *is a time-ordered list of application-level messages* $(t_n < t_{n+1})$ *exchanged over a network channel established between two devices that use a specific communications protocol.*

Because the Modbus, MMS and IEC104 protocols have different communications types (e.g., synchronous vs. asynchronous) and communications patterns (e.g., pushing vs. polling), Definition 1 is redefined for each protocol. Note that the term m is substituted by the sequence event e that details the properties and attributes of each protocol.

Definition 2: *A sequence of Modbus events is a time-ordered list of events* $\{e_{t_n}\}$ *where e is a three-tuple <ID, Code, Data> derived from two messages* $(m^{Req}{}_{t_n}, m^{Res}{}_{t>t_n})$.

Note that *ID* denotes the transaction identifier, *Code* indicates the type of operation performed and *Data* represents the information carried by Modbus requests m^{Req} and responses m^{Res}.

Definition 3: *A sequence of MMS events is a time-ordered list of events* $\{e_{t_n}\}$ *where e is a four-tuple <ID, PDU, Service, Data> derived from two messages* $(m^{Req}{}_{t_n}, m^{Res}{}_{t>t_n})$ *in the case of synchronous communications, and from* m_{t_n} *in the case of asynchronous communications.*

Note that *ID* denotes the invoke identifier, *PDU* indicates the type of communications (e.g., initiate, request, response, error, etc.), *Service* describes the operation requested or performed (e.g., read, write, etc.) and *Data* represents the information carried by MMS requests and responses.

Definition 4: *A sequence of IEC104 events is a time-ordered list of events* $\{e_{t_n}\}$ *where e is a three-tuple <Format, Service, Data> derived from two messages* $(m^{Req}{}_{t_n}, m^{Res}{}_{t>t_n})$ *in the case of synchronous communications, and from* m_{t_n} *in the case of asynchronous communications.*

Note that *Format* denotes the format of the messages, *Service* defines the performed service and *Data* represents the information carried by IEC104 messages.

5. Modeling Message Sequences

Section 4 described how to transform network traffic traces into time-ordered lists of events. The next step involves the modeling of message sequences to perform communications analysis and identify sequence attacks. A discrete-time Markov chain (DTMC) is used to model communications patterns and protocol behaviors. The modeling is done for two reasons:

- First, a flexible definition of event is needed that does not necessarily consider all the attributes used to build the sequence (e.g., ID, $Code$, $Data$). For example, two Modbus events that only differ in their transaction identifier (ID) are considered to be the same event because a

Algorithm 1 : DTMC modeling of sequences.

```
1: for all e_{t_n} ∈ sequence do
2:     State_{DTMC} ← extractAttributes(e_{t_n});
3:     if State_{DTMC} ∈ DTMC then
4:         update(State_{DTMC});
5:     else
6:         add(State_{DTMC}, DTMC);
7:     end if
8:     if Transition_{previousState,State_{DTMC}} ∈ DTMC then
9:         update(Transition_{previousState,State_{DTMC}});
10:    else
11:        add(Transition_{previousState,State_{DTMC}}, DTMC);
12:    end if
13:    previousState ← State_{DTMC};
14: end for
```

transaction identifier does not determine the meaning of a message or the significance of an event itself. DTMC states are used as sets of sequence events that share the same semantic meaning.

- Second, it is necessary to identify temporal consequent events. DTMC transitions are used to: (i) indicate the strength of the relationship between an event and its successor (e.g., how many times a state follows another); and (ii) understand if the relationship changes over time (e.g., the time interval between two states remains constant over time). A transition between two states A and B indicates an "episode of consequentiality" between two events belonging to A and B, respectively.

Modeling industrial control system communications with DTMCs has some advantages over using other modeling techniques such as n-grams or deterministic finite automata. Analysis using n-grams requires communications to be split into subsequences of messages of length n. Therefore, a model of industrial control system communications would be defined by the statistical distribution of the n-grams included in the entire sequence of messages. The resulting analysis would fail to identify subsequences of messages larger than n that remain the same during the entire communications. Modeling communications using a deterministic finite automaton would allow sequences of any length to be identified, but it would not be suitable for stochastic events (e.g., message delays). Without considering the probabilities of event occurrence, it would be impossible to evaluate the importance of transition functions and, consequently, to assess the correct behavior of industrial control system communications.

The construction of a DTMC is independent of the modeled protocol and is completely automated. From an implementation point of view, the algorithm used to build a DTMC reads a sequence of events one by one and populates the model. For every event in the sequence, the algorithm either assigns it to a

state or creates a new state if the event does not match the attributes of a state already in the model. Moreover, in each step, the algorithm adds or updates a transition function that links the previously-visited state to the current state. Algorithm 1 formalizes the DTMC modeling process.

Every state S is defined by a five-tuple $<Data, Type, \#Events, FTS, LTS>$ where *Data* denotes the information carried by events stored in S, *Type* indicates the type of events included in S (e.g., requests and responses, asynchronous requests), *#Events* denotes the number of events included in S, *FTS* (first time seen) is the timestamp of the first event in S and *LTS* (last time seen) is the timestamp of the last event in S.

Every transition T from a source state A to a destination state B is defined by a six-tuple $<TP, \#Jumps, FJ, LJ, ATE, \sigma ATE>$ where *TP* (transition probability) is the ratio of the number of jumps from A to B to the total number of jumps from A to any other state in the DTMC, *#Jumps* represents the number of jumps from A to B, *FJ* (first jump) is the timestamp of the first jump in T, *LJ* (last jump) is the timestamp of the last jump in T, *ATE* (average time elapsed) is the average time between two consequent events of A and B, and σATE is the standard deviation over all the intervals between the two consequent events of A and B.

To illustrate how the DTMC modeling process works, consider the following sequence of events:

1. **MMS Initiate Request/Response Event**: Invoke ID = –, PDU = Initiate, Service = –, Data = mmsInitRequestDetails, Timestamp = 15 Jun 2014 17:14:12.79
2. **MMS Confirmed Request/Response Event**: Invoke ID = 1, PDU = Confirmed Request/Response, Service = write (5), Data = octet-string (9) 00, Timestamp = 15 Jun 2014 17:14:12.973
3. **MMS Confirmed Request/Response Event**: Invoke ID = 2, PDU = Confirmed Request/Response, Service = write (5), Data = octet-string (9) 00, Timestamp = 15 Jun 2014 17:14:13.059
4. **MMS Confirmed Request/Response Event**: Invoke ID = 3, PDU = Confirmed Request/Response, Service = write (5), Data = octet-string (9) 00, Timestamp = 15 Jun 2014 17:14:13.311

Figure 1 shows the DTMC obtained by applying the modeling algorithm. The first event of the sequence creates a DTMC state of type Initiate Request/Response representing MMS initialization messages (state A). The second event of the sequence creates another state in the model of type Confirmed Request/Response that represents MMS messages used to write an octet-string at 00 (state B). At this point, the model also has a transition between the two states that describes the sequential nature of the two events of the sequence (Transition 1). The third event of the sequence has the same attributes as the second event and, thus, increases the attribute *#Events* of state B. However, a new transition connects state B in a self-loop to show the new relationship observed in the sequence. Finally, the fourth event of the sequence is still part of

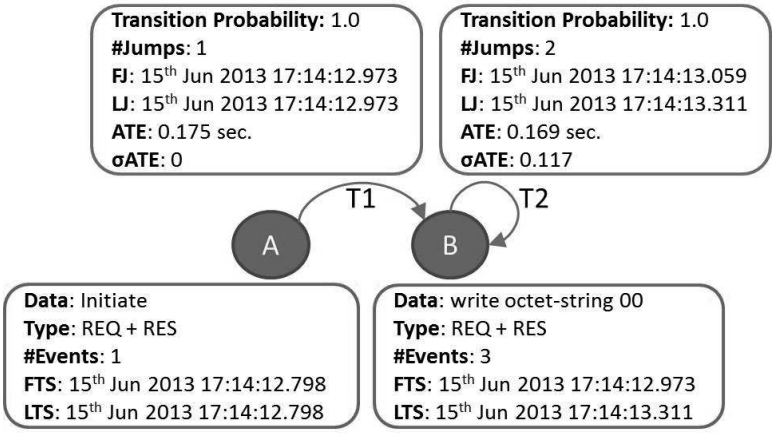


Figure 1. DTMC modeling algorithm example.

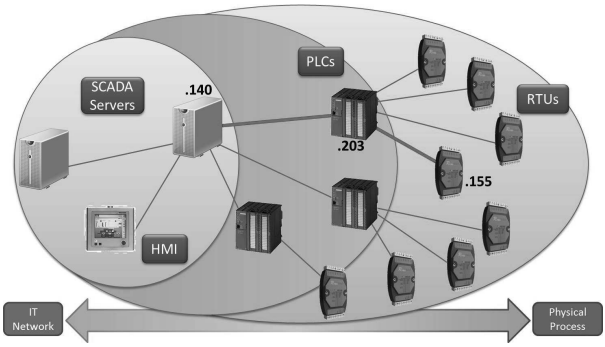
state B because it has the same attributes as the second event of the sequence. Since the model already has a transition that links state B in a self-loop, the algorithm only increases the *#Jumps* attribute of this transition.

6. Experiments and Analysis

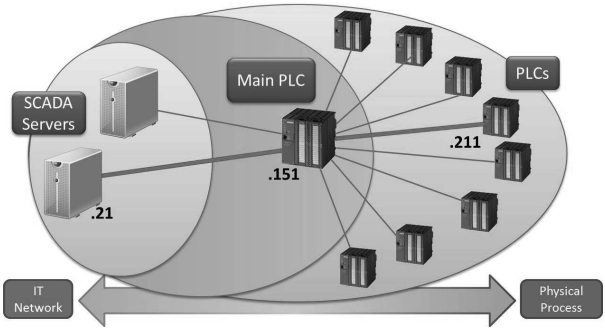
This section analyzes the DTMCs generated from real Modbus, MMS and IEC104 traffic. The network traces were captured at three utilities. The Modbus traffic was obtained from the control network of a water treatment plant. The MMS traffic was obtained from a gas storage and pipeline infrastructure; log files of the SCADA servers at this facility were also obtained. The IEC104 traffic was obtained from a gas distribution system and refers to secondary substations for distribution.

The timeframes of the data range from one day (MMS) to five days (Modbus and IEC104). It is worth noting that, during the traffic captures, no constraints were imposed on operations. All three infrastructures were running normally and operators were free to perform their tasks. The three samples represent the most realistic use cases in which a security system could be deployed, tested and tuned.

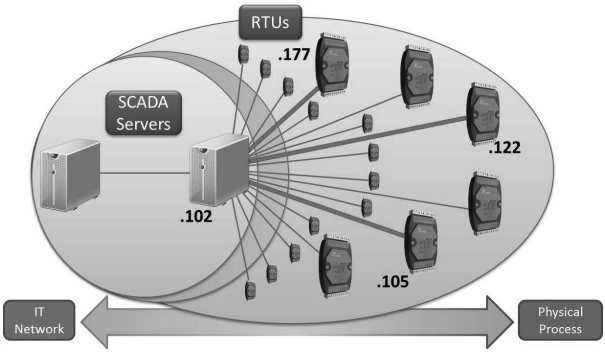
Figure 2 shows the topologies of the three networks tested. Sequencing and modeling operations were implemented on top of `Tshark` parsing services, which eliminated problems related to TCP stream reconstruction (e.g., retransmission and segment reordering). The `Tshark` output for a specific TCP stream was forwarded to a custom tool that selected the information needed to transform a message in a sequence event, and concatenated events to create sequences. At the same time, the modeling algorithm constructed the DTMC representing the sequence.



(a) Modbus network topology.



(b) MMS network topology.



(c) IEC104 network topology.

Figure 2. Network topologies.

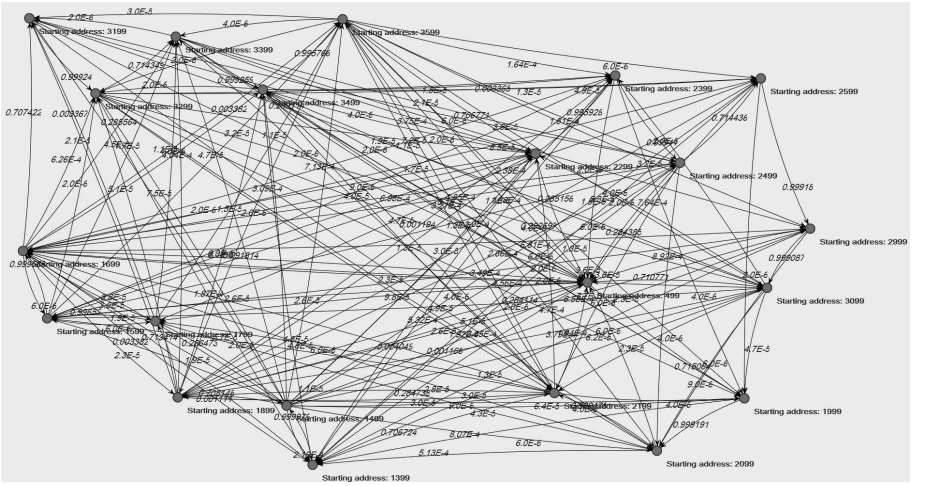


Figure 3. Modbus communications between SCADA server 140 and PLC 203.

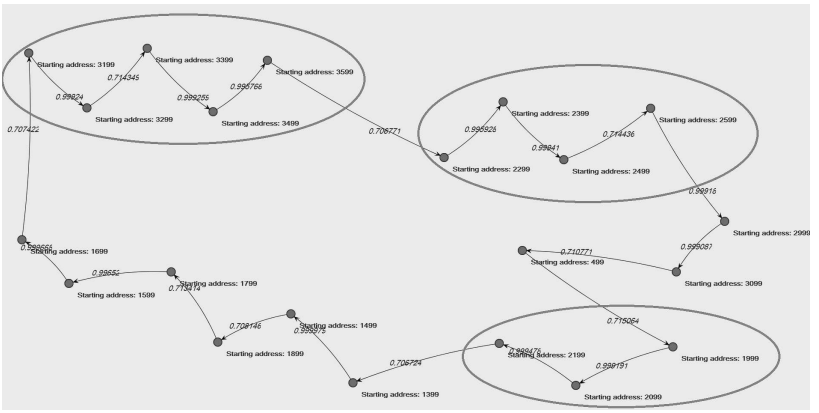
6.1 Modbus

In the Modbus experiment, 47 GB of traffic and more than 260M frames were captured. Every communication between a programming logic controller and remote terminal unit involved one Read Holding Register message sent in a loop. For this reason, the analysis focused on the connections between the SCADA server and the programming logic controllers. These TCP streams contained about 43M frames and each related sequence contained about 10M events. The generated models consisted of 21 to 22 states and a variable number of transitions (between 172 and 291). As mentioned above, Read Holding Register was the only message used, each message had a different set of addresses.

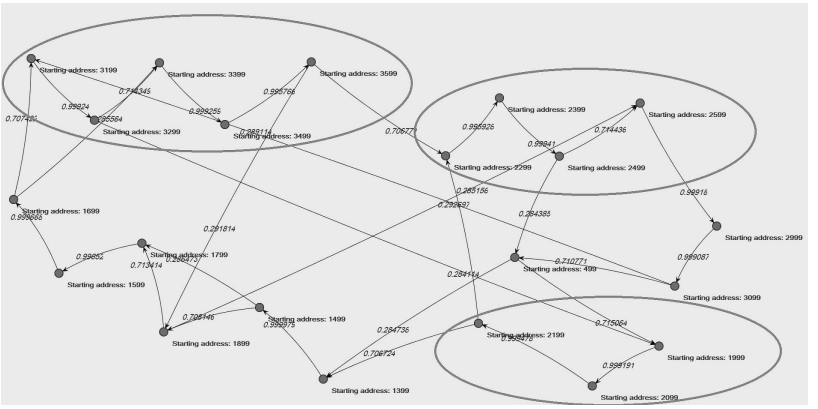
Figure 3 shows the Modbus communications between SCADA server 140 and programmable logic controller 203. Note that the models do not present the sequences of states with periodic behavior. However, the analysis of the DTMCs revealed the following outcomes:

- Given the total number of possible transitions within a DTMC (calculated by summing the possible edges $\#States \times (\#States - 1)$ and self-loops $\#States$), the observed transitions cover 40% to 60% of the domain. This means that, on average, half of the transitions did not occur even after five days of analysis.

- Not all the transitions have the same probability. Figure 4 shows a link with a transition probability higher than 0.7. Every involved transition has more than 150K jumps. The results indicate that hidden sequentiality exists even if it is disturbed by random delays. Only one-third of the states have two transitions with probability higher than 0.25 (more than



(a) DTMC transitions with probability > 0.7.



(b) DTMC transitions with probability > 0.25.

Figure 4. Modbus communications between SCADA server 140 and PLC 203.

130K jumps) and the 30 transitions shown in Figure 5 cover 99.8% of the jumps.

- No self-loops exist. None of the states in the model has a transition that goes back to itself. This means that no Read Holding Register operation was performed twice in a row.

- The transition distribution in the models is not uniform. Some states have higher numbers of transitions than others.

Moreover, it was possible to identify clusters of states (see Figure 4) with two properties: (i) each cluster has several edges that connect to states belonging

to other clusters; and (ii) each cluster has a few edges that connect internal states to each other and, among these edges, there is a path that connects all the states of the cluster and that contains almost all the jumps performed in the cluster (∼99.9998%).

These clusters denote the presence of several threads in the SCADA server operating system process (one per cluster) that multiplex requests in the same TCP stream. The task of each thread is to ask for a specific interval of registers. This hypothesis explains the difficulty in finding clear sequences (each thread can be randomly scheduled by the CPU) and the absence of some transitions (threads are always created in a loop and operations within the same thread are likely to be sequential).

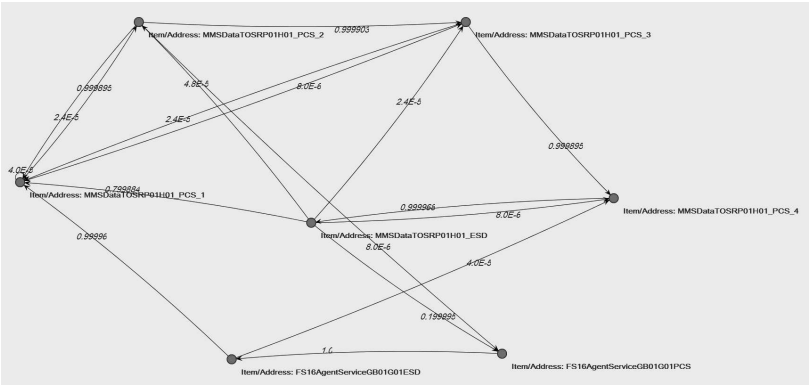
6.2 MMS

In the MMS experiment, 4 GB of traffic and more than 18.3M frames were captured. TCP streams that connected the main programmable logic controller (i.e., the device that coordinated all the field devices) to the other programmable logic controllers involved on average 2M frames. The generated sequences consisted of about 600K events with models containing two to seven states. The DTMCs had variable numbers of transitions that covered 35% to 75% of all the possible edges in the graphs. Models with the highest percentages of transitions consisted of TCP streams with only Read Request messages that were used to read two variables in the field programmable logic controllers. These models had three transitions (one self-loop and two transitions connecting the two states in each model). Models with higher numbers of states were suitable for sequence modeling in most of the cases.

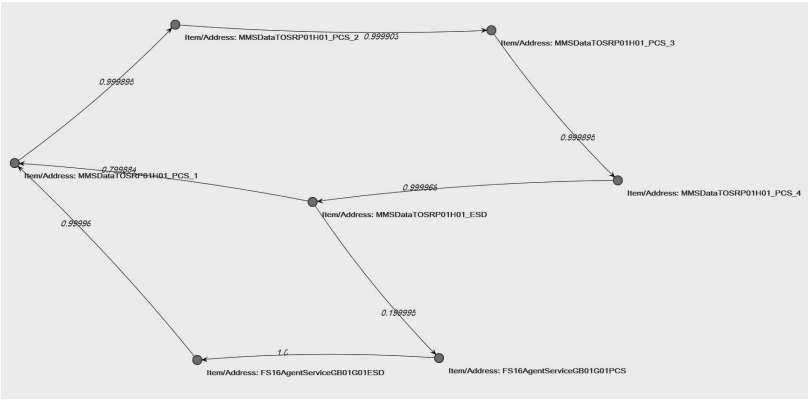
Figure 5(a) shows the communications model for the main programmable logic controller and a field programmable logic controller. The model has seven states and seventeen transitions. As in the Modbus experiments, there is a path that involves eight of the seventeen transitions; this path covers 99.99% of the jumps. Unlike the Modbus models, the MMS models show instances of strict consequentiality (e.g., between the two read operations at the bottom of Figure 5(a)). Although it is not known if altering the order of these two messages causes problems to the control process, it appears that their sequentiality is enforced by the system and, for this reason, it is likely to remain the same unless the control process changes.

The TCP streams that link the two SCADA servers to the main programmable logic controller include almost the same number of frames (1.9M) and sequence events (570K). This data is comparable with the MMS TCP streams discussed above. However, the related models change completely. The two new models include about 280 states and 1,280 transitions (see Figure 6). This is due to two reasons:

- The number of different operations performed among the SCADA servers and the main programmable logic controller is higher than the number of operations performed among the programmable logic controllers.



(a) Complete DTMC.



(b) DTMC transitions with probability > 0.19.

Figure 5. MMS communications between the main PLC 151 and PLC 211.

- The states in the new models mostly represent Read Unconstrained Address operations and the implementation of unconstrained addresses is proprietary. This causes every state to be identified by a byte string and, as a consequence, only events that share the exact same information are grouped together.

The correct parsing of Read Unconstrained Address operations would reduce the number of states. However, it is worth noting that:

- The percentage of performed transitions corresponds to 0.02% of all possible transitions. In the case of Modbus, the definition of a set of allowed sequences would be much more restrictive.

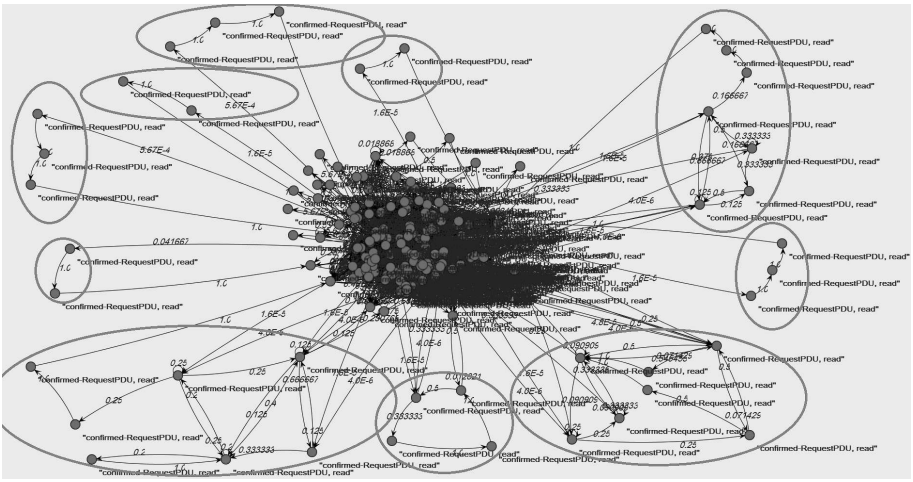


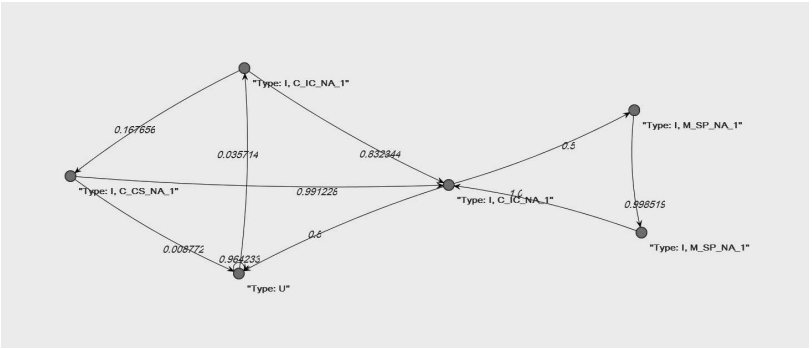
Figure 6. MMS communications between SCADA server 21 and PLC 151.

- About 45% of the states have "univocal relationships" between two other states (a state Y has a univocal relationship between states X and Z if transitions from X to Y are always followed by transitions from Y to Z; moreover, these states are often chained together). Such structures highlight subsets of MMS messages with strong consequentiality (i.e., subsets of messages that always follow each other).

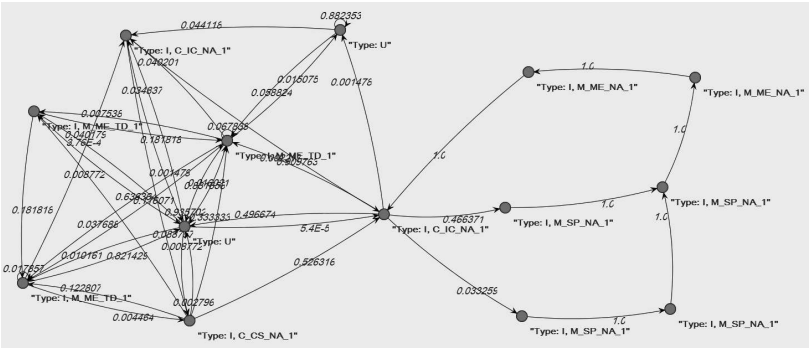
- The model yields several clusters of states (see Figure 6). More precisely, there is a densely-connected core with some smaller groups of states at the edges. Most of the little clusters are linked to the core by a few transitions that often involve only one jump.

  These clusters are the measurable effects of human intervention. This hypothesis is supported by the analysis of the process log file. First, human operations timestamps were matched against the starting times of the transitions. About 50% of the entries labeled as human operations were recorded in the same time frames as when the transitions occurred. Second, the time frames with no human operations were examined; almost the same time gaps were observed for the transitions. Finally, it was observed that the two time frames with the highest numbers of transitions of this kind corresponded to the only two operations recorded as "suppress" in the log file. According to plant operators, these operations correspond to the manual termination of system alarms with a consequent reset of several system control parameters to the "normal" state.

The analyses of Modbus and MMS traffic samples emphasize the differences between programmable logic controller to programmable logic controller (or remote terminal unit to remote terminal unit) and server to programmable



(a) SCADA server 102 and RTU 122 communications.



(b) SCADA server 102 and RTU 177 communications.

Figure 7. IEC104 communications.

logic controller (or server to remote terminal unit) communications. The models demonstrate that the former type of communications exhibits fewer states and transitions while the later usually has a greater variety of messages.

6.3 IEC104

In the IEC104 experiment, 51 GB of traffic and more than 203 M frames were captured. TCP streams linking the SCADA server and the remote terminal units were analyzed. These connections contained 70K to 96K frames and 13K to 24K sequence events. As in the MMS experiments, the number of sequence events did not depend on the number of frames.

Most of the models had a number of transitions that covered 20% to 33% of all possible transitions. Figures 7(a) and 7(b) illustrate this case with TCP streams of about 24K events. Figure 7(a) shows a model with six states and

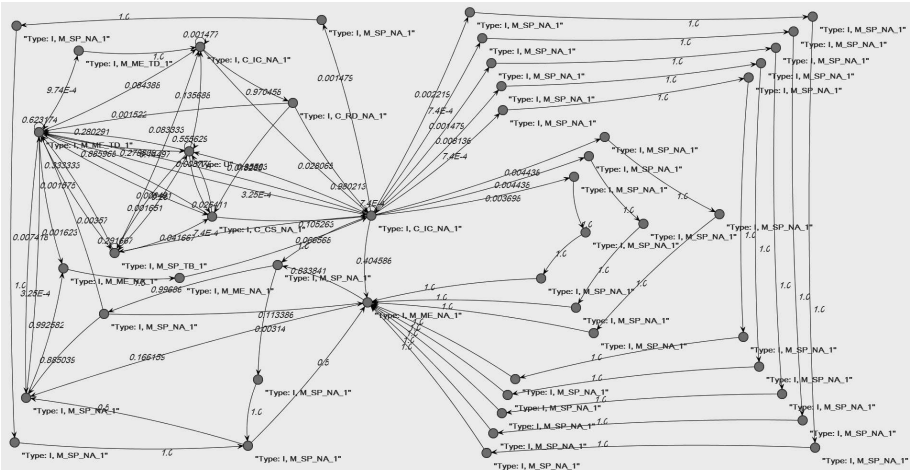


Figure 8. IEC104 communications between SCADA server 102 and RTU 105.

ten transitions while Figure 7(b) shows a model with fourteen states and 43 transitions. Both the models have two sets of states with different densities of transitions. The group of states on the left involves different types of commands and is responsible for the majority of transitions that do not show sequentiality. The other group of states on the right involves "single point information" commands and shows some clear paths. As discussed in the case of MMS, this portion of the model allows some assumptions to be made regarding state sequences (e.g., consequentiality of single point information messages).

The model in Figure 8 corresponds to another server to remote terminal unit TCP stream with fewer events (13K), but with a greater number of states (49) and transitions (92). The percentage of transitions out of the total number of possibilities is lower than in the previous cases (around 0.04%). The model shows also several precise paths. The IEC104 results lead to the following outcomes:

- All the examples demonstrate specific parts of the model with no sequentiality. Messages that create these clusters are almost the same in the three cases (e.g., TESTFR functions, Measured Value – Normalized Value commands).
- The remaining part of the model includes several definite sequences. The analysis reveals that, despite the presence of different paths, the involved states all relate to a precise sequence of addresses to be read. Every state collects single point information commands with a different number of object addresses. The set of object addresses at the end of every path is always the same as well as the order in which they are used.

Finally, it is finally worth noting that the analysis of the IEC104 traffic reveals a lower number of events compared with Modbus and MMS traffic. This

is mainly due to long delays for several messages. Furthermore, the standard deviation of the related transitions is higher. This result can be explained by the IEC104 pushing pattern of communications, which reduces the number of messages in the network and makes the transition timings less precise. This behavior can complicate sequence analysis for the two types of misuse discussed in Section 3.

6.4 Discussion

Despite the stability of industrial control system communications (e.g., long-running TCP sessions and constant patterns) [6], this research highlights some challenges to sequence analysis. In most of the cases, the generated models do not present definite paths (only the IEC104 results show some precise sequences). Moreover, the MMS server to programmable logic controller communications demonstrate the effect that a larger set of different messages has on the numbers of states and transitions. Nevertheless, certain properties that should be considered when developing sequence-aware intrusion detection systems were identified.

All the models produce numbers of edges that are much lower than the maximum permitted by the graphs. This result suggests that messages cannot be combined to create every possible sequence, but that only specific sequences are permitted. This is a necessary condition for sequence-aware intrusion detection. Furthermore, the analysis shows that only a small fraction of the transitions are used most frequently. This information emerges from the models by filtering out transitions with few jumps. The remaining transitions often form precise paths through the states and, thus, strengthen the hypothesis of substantial sequentiality in the communications. The sequentiality can be weakened by random delays, but this does not compromise the operations of a sequence-aware intrusion detection system.

From this analysis, it is possible to envision two different, yet interoperable, sequence detection mechanisms. The first mechanism enforces sequentiality constraints defined by the referenced models (e.g., by observing the univocal relationships defined for MMS). The second focuses on the probability distributions of the constraints (e.g., requiring the relative probabilities of the transitions between a state and its neighbors to remain the same).

The first mechanism could be used to detect the "order-based" attacks described in Section 3, because these attacks involve observing a sequence that should not be found (i.e., a sequence that is not present in the model constructed using normal (attack-free) traffic). The second mechanism could be used to detect "timing-based" attacks. Indeed, these attacks involve sequences allowed by the model that occur much more frequently compared with the normal. In this case, the DTMC would allow the transitions until the probability of choosing a path exceeds the one computed based on normal traffic.

Finally, some observations must be made regarding the relationships between transitions that involve low numbers of jumps and human operations. MMS traffic samples together with the process log file permit the extraction of tem-

poral matches. Certain correlations were observed and the analysis supports the assumption that links human interactions to such transitions and, specifically, to some of the clusters shown in Figure 6. This supports the profiling of operator actions and differentiating them from malicious activities. Additional experimentation and analysis would be valuable for sequence-aware intrusion detection because they would decrease the number of false positives by helping filter out sequences that are known to be operator actions.

7. Related Work

Protocol sequence analysis and modeling are activities that are usually related to communications-model-based verification and validation. Aarts et al. [1] have presented an example of this approach. They implemented a regular inference technique based on Mealy machines and tested it on the SIP and TCP protocols. Then, a predicate abstraction framework was used to infer finite state models of network components from observations of external behavior.

Chandola et al. [2] have presented three formulations of the sequence anomaly detection problem: (i) identifying anomalous sequences with respect to a set of known normal sequences; (ii) identifying anomalous subsequences within long sequences; and (iii) identifying subsequences whose frequencies of occurrence are anomalous. They identified suitable approaches for each formulation and detailed their advantages and disadvantages.

A search of the literature reveals that few examples of sequence-aware network intrusion detection systems exist. Sekar et al. [22] have proposed a specification-based modeling approach based on extended finite state automata. They demonstrated the feasibility of analysis based on well-defined network protocols such as UDP and TCP. However, they did not test their approach on application layer protocols and did not investigate the possibility of automatically learning communications patterns in cases where comprehensive specifications are not available.

Krueger et al. [14] went beyond the approach of Sekar et al., using n-grams and Markov chains to model protocols such as SIP, DNS and FTP. They demonstrated the effectiveness of the detection mechanisms on text protocols, but they did not test them against binary protocols. Industrial control systems mainly use binary protocols and studies such as [7] highlight the problems of using n-gram analysis in industrial control environments.

Some researchers have attempted to address this issue [5, 26]. Goldenberg and Wool [5] used a deterministic finite state automaton to create a model from real Modbus traffic. Their system identifies communications channels for each pair of devices that communicate with each other (e.g., human-machine interface and programmable logic controller, and programmable logic controller and programmable logic controller). After the channel is set, the system records the permitted transitions of the deterministic finite state automaton by examining the sequence of Modbus messages within the channel. During the detection phase, every unexpected transition is flagged as an anomaly. Anomalies are

of three types: retransmission (occurrence of a deterministic finite state automaton symbol that is the same as the previous symbol), miss (occurrence of a known symbol in an unexpected position) and unknown (appearance of an unknown symbol). Goldenberg and Wool rely on the assumption that Modbus traffic is highly periodic. However, the tests conducted in this research have demonstrated that Modbus traffic is periodic only if random delays are filtered from the model. The approach of Goldenberg and Wool, if applied to the Modbus traffic samples used in this work, would have been inadequate to model communications and would have created an unmanageable number of automaton states and a large number of false positives.

Yoon et al. [26] have also focused on Modbus, but they modeled communications using dynamic Bayesian networks and probabilistic suffix trees. Each communications channel is reduced to a sequence of elements by parsing Modbus messages and pairing requests and responses. The given sequence is fed to the probabilistic suffix tree model and becomes the dataset that is used to detect anomalous communications. In the detection phase, the system evaluates every new sequence of messages by considering the likelihood of generating the sequence using the probabilistic suffix tree model. The system triggers an alert if the result is below a threshold. Yoon et al. tested their intrusion detection system on traffic generated by a testbed as well as on a synthetic data trace. However, they relied on the predictability of industrial control system traffic. Yoon et al. also implemented a mechanism to deal with false positives caused by single missing messages. This mechanism evaluates the probability that an anomalous sequence is missing a message (e.g., due to network delays). The mechanism considers a sequence to be normal traffic if the action of restoring the message allows the sequence to be accepted by the probabilistic suffix tree model, and the detection process proceeds to the next sequence. However, although the mechanism handles most of the false positives, Yoon et al. did not consider the impact on false negatives (e.g., caused by attacks that opportunistically filter specific network messages).

Finally, it is worth mentioning that there are several examples of sequence-aware, host-based intrusion detection systems. Most of these systems profile program activities and system calls (see, e.g., [8, 17, 25]) while other systems focus on user activities (see, e.g., [15]).

8. Conclusions

This chapter has presented a methodology for modeling and analyzing industrial control system communications. The methodology models sequences of messages as discrete time Markov chains (DTMCs). This is accomplished by extracting information from network frames. An algorithm is then used to model specific features of the communications. Finally, DTMCs are used to understand the communications patterns and the consequentiality among messages. Two sequence detection mechanisms have been proposed based on the information provided by the DTMCs. The first is a deterministic approach

that enforces specific sequentiality constraints. The second is a probabilistic approach that exploits transition probability distributions.

Future research will focus on developing a sequence-aware intrusion detection system. Research will also analyze other industrial control system protocols such as DNP3 and Profinet to expand the variety of communications patterns that can be handled, with the ultimate goal of developing high-performance intrusion detection systems for industrial control environments.

Acknowledgements

This research was partially supported by the European Commission via Project FP7-SEC-285477-CRISALIS of the 7th Framework Programme [3]. This research was also supported by a CTVR Grant (SFI 10/CE/I 1853) from Science Foundation Ireland.

References

[1] F. Aarts, B. Jonsson and J. Uijen, Generating models of infinite-state communications protocols using regular inference with abstraction, *Proceedings of the Twenty-Second IFIP WG 6.1 International Conference on Testing Software and Systems*, pp. 188–204, 2010.

[2] V. Chandola, A. Banerjee and V. Kumar, Anomaly detection for discrete sequences: A survey, *IEEE Transactions on Knowledge and Data Engineering*, vol. 24(5), pp. 823–839, 2012.

[3] CRISALIS Project, CRISALIS – Securing Critical Infrastructures, Siemens, Munich, Germany (`www.crisalis-project.eu`), 2012.

[4] N. Falliere, L. O'Murchu and E. Chien, W32.Stuxnet Dossier, version 1.4, Symantec, Mountain View, California, 2011.

[5] N. Goldenberg and A. Wool, Accurate modeling of Modbus/TCP for intrusion detection in SCADA systems, *International Journal of Critical Infrastructure Protection*, vol. 6(2), pp. 63–75, 2013.

[6] D. Hadziosmanovic, D. Bolzoni, S. Etalle and P. Hartel, Challenges and opportunities in securing industrial control systems, *Proceedings of the Workshop on Complexity in Engineering*, 2012.

[7] D. Hadziosmanovic, L. Simionato, D. Bolzoni, E. Zambon and S. Etalle, N-gram against the machine: On the feasibility of n-gram network analysis for binary protocols, *Proceedings of the Fifteenth International Symposium on Research in Attacks, Intrusions and Defenses*, pp. 354–373, 2012.

[8] S. Hofmeyr, S. Forrest and A. Somayaji, Intrusion detection using sequences of system calls, *Journal of Computer Security*, vol. 6(3), pp. 151–180, 1998.

[9] Industrial Control Systems Cyber Emergency Response Team, Advisory (ICSA-14-073-01), Siemens SIMATIC S7-1500 CPU Firmware Vulnerabilities, Department of Homeland Security, Washington, DC (`ics-cert.us-cert.gov/advisories/ICSA-14-073-01`), March 17, 2014.

[10] International Electrotechnical Commission, IEC 60870-5-101, Telecontrol Equipment and Systems – Part 5-101: Transmission Protocols – Companion Standard for Basic Telecontrol Tasks, Geneva, Switzerland, 2003.

[11] International Electrotechnical Commission, IEC 60870-5-104, Transmission Protocols, Network Access for IEC 60870-5-101 Using Standard Transport Profiles, Geneva, Switzerland, 2006.

[12] International Organization for Standardization, ISO 9506-1: Industrial Automation Systems – Manufacturing Message Specification, Part 1: Service Definition, Geneva, Switzerland, 2003.

[13] International Organization for Standardization, ISO 9506-2: Industrial Automation Systems – Manufacturing Message Specification, Part 2: Protocol Specification, Geneva, Switzerland, 2003.

[14] T. Krueger, H. Gascon, N. Kramer and K. Rieck, Learning stateful models for network honeypots, *Proceedings of the Fifth ACM Workshop on Security and Artificial Intelligence*, pp. 37–48, 2012.

[15] T. Lane and C. Brodley, Sequence matching and learning in anomaly detection for computer security, *Proceedings of the AAAI-97 Workshop on AI Approaches to Fraud Detection and Risk Management*, pp. 43–49, 1997.

[16] R. Langner, To Kill a Centrifuge: A Technical Analysis of What Stuxnet's Creators Tried to Achieve, The Langner Group, Arlington, Virginia, 2013.

[17] G. Mao, J. Zhang and X. Wu, Intrusion detection based on the short sequence model, *Proceedings of the Seventh World Congress on Intelligent Control and Automation*, pp. 1449–1454, 2008.

[18] S. McLaughlin and P. McDaniel, SABOT: Specification-based payload generation for programmable logic controllers, *Proceedings of the ACM Conference on Computer and Communications Security*, pp. 439–449, 2012.

[19] Modbus Organization, Modbus Application Protocol Specification (v1.1a), Hopkinton, Massachusetts (`www.modbus.org/specs.php`), 2004.

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

[21] President's Commission on Critical Infrastructure Protection, Critical Foundations: Protecting America's Infrastructures, The Report of the President's Commission on Critical Infrastructure Protection, The White House, Washington, DC, 1997.

[22] R. Sekar, A. Gupta, J. Frullo, T. Shanbhag, A. Tiwari, H. Yang and S. Zhou, Specification-based anomaly detection: A new approach for detecting network intrusions, *Proceedings of the Ninth ACM Conference on Computer and Communications Security*, pp. 265–274, 2002.

[23] K. Stouffer, J. Falco and K. Scarfone, Guide to Industrial Control Systems (ICS) Security, NIST Special Publication 800-82, National Institute of Standards and Technology, Gaithersburg, Maryland, 2011.

[24] A. Swales, Open Modbus/TCP Specification, Release 1.0, Schneider Electric, Rueil-Malmaison, France, 1999.

[25] C. Warrender, S. Forrest and B. Pearlmutter, Detecting intrusions using system calls: Alternative data models, *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 133–145, 1999.

[26] M. Yoon and G. Ciocarlie, Communication pattern monitoring: Improving the utility of anomaly detection for industrial control systems, presented at the *NDSS Workshop on Security of Emerging Networking Technologies*, 2014.