Sequence-aware Intrusion Detection in Industrial Control Systems*

Marco Caselli
University of Twente
m.caselli@utwente.nl

Emmanuele Zambon
University of Twente & SecurityMatters BV
emmanuele.zambon@secmatters.com

Frank Kargl
University of Twente & Ulm University
frank.kargl@uni-ulm.de

ABSTRACT

Nowadays, several threats endanger cyber-physical systems. Among these systems, industrial control systems (ICS) operating on critical infrastructures have been proven to be an attractive target for attackers. The case of Stuxnet has not only showed that ICSs are vulnerable to cyber-attacks, but also that some of these attacks rely on understanding the processes beyond the employed systems and using such knowledge to maximize the damage. This concept is commonly known as "semantic attack". Our paper discusses a specific type of semantic attack involving "sequences of events". Common network intrusion detection systems (NIDS) generally search for single, unusual or "not permitted" operations. In our case, rather than a malicious event, we show how a specific series of "permitted" operations can elude standard intrusion detection systems and still damage an infrastructure. Moreover, we present a possible approach to the development of a sequence-aware intrusion detection system (S-IDS). We propose a S-IDS reference architecture and we discuss all the steps through its implementations. Finally, we test the S-IDS on real ICS traffic samples captured from a water treatment and purification facility.

Categories and Subject Descriptors

C.2 [**COMPUTER-COMMUNICATION NETWORKS**]: General—*Security and protection*

General Terms

Cyber-physical system; Intrusion detection system; Semantic attack; Sequence attack

*This work has been partially supported by the European Commission through project FP7-SEC-285477-CRISALIS funded by the 7th Framework Program.

CPSS'15, April 14, 2015, Singapore.

ACM 978-1-4503-3448-8/15/04 ...$15.00.
http://dx.doi.org/10.1145/2732198.2732200.

1. INTRODUCTION

Industrial control system (ICS) is a term that refers to a large and broad variety of systems used in industry for monitoring purposes. These systems have been used for years, but they have recently acquired new functionalities and are now more similar to classical networked IT systems. Infrastructures that used to be standalone entities have become interconnected and remotely accessible by operators. This change improves the manageability of industrial control systems but it also increases their exposure to cyber-threats. A cyber-attack that strikes ICSs can endanger the infrastructure managed by the control system. In case of critical infrastructure (e.g., power plant, electrical grid) the attack may also affect citizens and can put human lives at risk.

Not all cyber-threats endanger ICSs in the same way. Attacks such as Denial of Service (DoS) strike the IT control infrastructure but do not directly target the controlled physical system. Other attacks instead rely on knowledge about the control processes or even the physical systems controlled by ICSs and try to maximize the damage inflicted on the physical world. This kind of attacks goes under the name of *semantic attacks*.

Semantic attacks require deep knowledge of protocols, software, hardware and physical systems involved in an infrastructure. The more an attacker knows about the targets the better he can trigger the systems into inconsistent or dangerous states. Semantic attacks can be of several types. Stuxnet [7] is one of the most known semantic attacks. The malware was designed to manipulate embedded software of Programmable Logic Controllers (PLCs) of Iranian nuclear enrichment facilities and disrupt nuclear centrifuges.

Our work focuses on a specific type of semantic attacks called *sequence attacks*. These attacks concern the misplacement of events within a sequence of ICS operations. Sequence attacks do not involve events that are malicious per se (e.g., unknown connections, out-of-range values of process variables, etc.). Instead, they exploit the possibility to arrange "valid" events (e.g. network messages, log entries, variable values) in a way that their presence, in relation with other operations, can cause problems to targeted devices (e.g., faults, failures).

Sequence attacks are not new in literature. A U.S. report on critical infrastructure protection describes a sequence attack being the cause of "water hammer effects" [24]. This attack relies on rapidly opening and closing control valves

in order to break them. Furthermore, Carcano e.a. [8] show the impact of a sequence attack on a pipe where the high pressure steam flowing on it is regulated by two valves. By closing and opening these valves with the right timing the authors succeed to increase the pressure to a critical value.

Common intrusion detection systems (IDSs) generally search for single events that either show clearly malicious or at least "unusual" characteristics. However, sequence attacks do not involve such characteristics and are likely to pass through unnoticed. An approach to anomaly-based intrusion detection, called "specification-based" detection, can solve specific cases of sequence attacks but its implementation relies on the presence of an exhaustive and accurate specification of system operations, like shown, for example, for some protocols of the TCP/IP protocol suite [22]).

In this paper we propose a different solution for sequence attack detection on ICS systems. Our approach relies on a sequence-aware intrusion detection system (S-IDS) that *identifies patterns of ICS network events, extracts their semantic meaning and models known behaviors over time*. We define a S-IDS reference architecture and show a prototypical implementation of our approach. In our work, we make use of discrete-time Markov chains (DTMCs) to describe several ICS device operations acquired by previously recorded network messages and log entries. Furthermore, we define a detection mechanism based on the computation of a weighted-distance among Markov chain states. Finally, we test our implementation on data and communication traces belonging to real ICS infrastructures.

The rest of the paper is structured as follows. Section 2 gives a brief overview of ICS and introduces the sequence attacks. Section 3 discusses intrusion detection state of the art. Section 4 proposes a reference architecture for a sequence-awere intrusion detection system. The implementation of the S-IDS is discussed in Sections 5, 6 and 7. We test the S-IDS on a real scenario in Section 8. Finally, Section 9 concludes the work and outlines future work.

2. INDUSTRIAL CONTROL SYSTEMS AND "SEQUENCE ATTACKS"

Given the number of different ICS deployments and the variety of possible threats endangering each one of those we narrow the scope of this paper to a basic set of ICS devices and cyber-attacks of interest.

2.1 ICS Overview

The acronym ICS encompasses a large variety of technologies such as Control and Data Acquisition Systems (SCADA), Distributed Control Systems (DCS), and Programmable Logic Controllers (PLCs) [23].

ICS networks gather together a number of different devices. The so called "field devices" are close or connected to the physical process under control. These devices include:

- *Sensors and Actuators*: components that directly measure and modify physical parameters
- *Remote Terminal Units (RTUs)*: electronic control devices that act as an interface to the physical process
- *Programmable Logic Controllers*: devices running the programs that instruct sensors and actuators and sending data back and forth to the process control network

Devices supervising the control process include:

- *SCADA Servers*: machines that manage and coordinate PLCs and RTUs
- *Human Machine Interfaces*: components that provide user-friendly interfaces to engineers
- *Engineering Workstations*: computers used to program PLCs

Every ICS system uses a set of communication protocols to exchange information and manage devices. Industrial control systems relied on serial communications for decades. In the last years, TCP/IP- and Ethernet-based networking technologies have been increasingly integrated in these infrastructures. For this reason, several industrial protocols have been ported to the TCP/IP protocol stack.

We mostly focus our research on three different protocols: Modbus, MMS and IEC 60870-5-104. These protocols are widely used in industrial control systems and include a significant variety of properties and characteristics.

- *Modbus*: is an application layer protocol, de facto standard for industrial systems [20]. Modbus communications rely on a client/server paradigm. In every communication, a client machine sends requests towards one or more PLCs and waits for responses.
- *MMS*: the Manufacturing Message Specification [14] implements all seven layers of the ISO/OSI stack. However, the most used version of the protocol works over TCP/IP. MMS is a client/server protocol with synchronous or asynchronous communication patterns.
- *IEC 60870-5-104*: is part of the IEC 60870 set of standards defining mechanisms used for telecontrol [6]. IEC 60870-5-104 defines the ISO/OSI application layer of the standard and works over TCP/IP. The protocol mostly uses asynchronous balanced or unbalanced data transfer modes. In the remainder of this paper, we will refer to IEC 60870-5-104 as "IEC104".

2.2 Sequence attacks

Our work focuses on "semantic attacks" and, particularly, aims to detect a specific kind of semantic attacks, namely "sequence attacks". We can divide sequence attacks into two different sub-sets: "order-based" and "time-based" attacks. The former are attacks in which messages or commands are sent with an incorrect/malicious order. The latter are attacks in which messages or commands are sent with an incorrect/malicious timing.

Carcano et. al. discuss in [8] an example of order-based sequence attack. Their setup involves a pipe where some steam flows at high pressure. Two valves (V1 and V2) control the pressure within the pipe. The authors show that it was possible for an attacker to put the system in danger by just misplacing two fully legitimate control messages. In fact, an attacker that has access to the network can inject a write message to the PLC controlling the valves to force V2's complete closure and V1's complete opening. These instructions have the effect to maximize the incoming steam and, thus, the pressure within the pipe. Each of these commands are perfectly legal when considered individually, while sending them in the specified order will bring the system to a critical state.

A report from the US President's Commission on Critical Infrastructure Protection [24] presents a similar example of a time-based sequence attack. The discussed scenario involves the water distribution sector. The authors of the report explain how water pipelines are controlled by major control valves. These valves can be rapidly opened and closed causing a so-called *water hammer effect*[1], which could result in a large number of simultaneous main breaks in the pipeline. Such an attack could be carried out by an attacker that sends an unusually rapid sequence of legitimate write messages issuing open and close commands to the PLCs controlling these major control valves.

A "sequence-aware" IDS has to deal both with order-based and time-based sequence attacks. In what follows we are going to present a detection mechanism aiming to effectively detect these kinds of attack.

3. RELATED WORK

According to [5] there are two main categories of network intrusion detection systems: *misuse-based* and *anomaly-based*. Misuse-based intrusion detection systems, also referred as signature-based, rely on a precise definition of malicious behavior (e.g., the description of a malicious network message or a more complex list of properties profiling attackers' activities). Anomaly-based intrusion detection systems use instead a complementary approach. These systems exploit a definition of normal behavior and flag everything that does not match the definition. Building a database of normal behaviors is an operation that usually requires a learning phase. In this phase, an anomaly-based intrusion detection system records information of the network with the assumption that no malicious activities are performed in the meantime, and extracts the parameters that define its knowledge of what is allowed or not. A different way to perform this operation, is by learning such normal behaviors from the specifications of a system or a protocol. This method (sometimes referred as a completely different category of intrusion detection systems, named "Specification-based" [22]) allows to build a model of normal behavior without learning phase.

The object of the analysis of a network intrusion detection system can be of different kinds. The most common type is the content of network messages.. Both misuse-based and anomaly-based intrusion detection systems can look into message headers or payloads to collect valuable information on which to base their analyses. Two examples of intrusion detection systems based on header analysis are the works proposed in [12] and [18]. Works described in [25, 2, 21] are examples of payload-based approaches. It is worth mentioning that Scheirer et al. [21] go beyond a static analysis of payloads by extracting binary codes and analyzing their semantic to model applications' behaviors. A further approach to intrusion detection relies on network flows analysis. Flow-based intrusion detection profiles network traffic in terms of connections or properties of the traffic (e.g., peeks, bursts, etc.) to identify unknown or malicious trends in the data, such as [15] and [1].

The approach we describe in this paper shifts the object of the analysis on sequences of messages. Chandola et al. discuss several theoretical approaches to sequence analysis and modeling in [4]. The authors provide three different formulations of the sequence anomaly detection problem: (1) identifying anomalous sequences with respect to a set of known normal sequences; (2) identifying anomalous subsequences within a long sequence; and (3) identifying subsequences whose frequencies of occurrence are anomalous.

To the best of our knowledge there are only few examples of S-IDSs implementations in literature. Sekar et al. [22] propose to profile IP, ARP, TCP and UDP communications in a network. The authors discuss the concept of specification-based anomaly detection and describe the steps to implement a detection system based on extended finite state automata (EFSA). This work proves the feasibility of an analysis based on comprehensive protocol specifications but it does not extend the approach to application-layer protocols. More importantly, it does not investigate the possibility to learn such patterns in case precise specifications are missing.

Krueger et al. [16] implement a method for protocol inspection and state machine analysis called PRISMA. Their approach involves n-grams to analyze message similarities and to group them into events and it uses Markov chains to link such clusters together and to identify rules defining the behavior of a network communication. [22] shows a method to model communication protocols by looking at the network traffic. However, the authors focus their research on text protocols and do not test their approach on binary protocols. Industrial control systems mainly use binary protocols and works such as [10] highlight the difficulties of using n-gram analysis in such environments.

Works such as [9, 27] attempt to answer to this last issue. Goldenberg et al. [9] focus on a industrial application protocol (Modbus) and use a deterministic finite automaton (DFA) to build a model from real traffic traces. The authors rely on the assumption that Modbus traffic is highly periodic and they argue that tests performed on real traffic show a low-rate of false positive. It is worth noting that the human interaction with the system is never taken into account in the modeling phase, causing every activity performed by an operator to be an anomaly of unknown type. Beyond, our tests show that Modbus traffic is periodic only if we are able to filter out random delays. When applying the approach proposed by Goldenberg et al. on our real-world Modbus traffic samples it shows to be insufficient to model the communications and would have created an unmanageable number of states in the DFA and a high number of false positive in the detection phase. So therefore, a more refined modeling approach is necessary.

Yoon et al. [27] also focus on Modbus but they model communications using dynamic Bayesian networks (DBNs) and probabilistic suffix trees (PSTs). When a new sequence is captured, the system looks at the likelihood to generate this sequence from the PST model. As discussed for [9], the authors rely on the predictability of industrial control system traffic. However, they implement a mechanism that evaluates whether an anomalous sequence is just missing a message (e.g., due to a delay on the network). The algorithm works as follows: if the action of restoring the message allows the sequence to be accepted by the PST, the system considers it as "normal traffic" and proceeds. Even if this mechanism removes most of the false positives, the authors do not discuss the impact on the number false negatives (e.g. real attacks opportunely filtering specific messages on the network) which we suspect to become a problem.

[1] A pressure wave generated within a fluid in motion due to sudden stops or changes of direction.

Hadžiosmanović et al. [11] approach the problem in a different way by looking directly at ICS process variables. The authors extract variable values from devices' network communications and use autoregression modeling and control limits to monitor their changes over time. When a value does not fit the model or exceeds the control limits, the intrusion detection system raises an alert providing the correct expected behavior. The approach shows promising results but faces several difficulties in modeling specific classes of variables (e.g. floating points). This is due to the way ICS applications deal with variables. When the size of a process variable exceeds the memory unit used by an application (e.g., 16 bits in Modbus) the value is split and saved within two or more memory locations. The intrusion detection system is unable to follow such mapping and treats every memory location independently from the others. This ultimately causes the usage of erroneous models.

Finally, it is worth mentioning that there are several examples of sequence-aware approaches to intrusion detection in the field of generic host-based intrusion detection systems. Some of this works focus on profiling user activities [17]. Others focus instead on profiling programs or sequences of system calls [13, 19, 26].

Having said this, we argue that sequence attack detection within industrial control systems still deals with the following limitations:

Security solutions such as intrusion detection systems cannot recognize semantic attacks without any knowledge of the infrastructure and the physical processes under control

Focusing on a large and comprehensive sets of characteristics can cause models to grow and become unmanageable

Random delays and human operations can create noise within models and can decrease detection accuracy

Our work attempts to address these three limitations.

4. S-IDS ARCHITECTURE

In Figure 1, we propose an architecture for a sequence-aware intrusion detection system which is based on a layered structure. Each layer receives information items from a lower layer, evaluates them, and finally forwards the results to the following layer. A layered structure allows to abstract from the input sources and to improve both usability a maintainability of the S-IDS.

The first layer is the *Reader*. The Reader is in charge of capturing raw information (e.g., files, network packets, data streams, etc.) and generates a uniform and identically formatted input stream for the S-IDS. Furthermore, the Reader filters out redundant or corrupted data that will not be needed by the following layers.

The *Sequencer* is the core of the architecture. This layer is in charge of transforming the data flowing through the Reader into a temporal sequence of events. This activity relies on a precise definition of what an event is. Depending on the goal of the analysis, the event is defined by a set of rules used by the Sequencer to organize such information in formalized data units. As an example, read messages may be considered one type of event while all write messages can form another type of event. If needed, a more fine-grained categorization of events can be implemented (e.g., also depending on the register read or written). Thereafter, the Sequencer sorts the units according to a timestamp representing the moment in which the event has occurred. Finally, the obtained sequence is forwarded to the next layer.

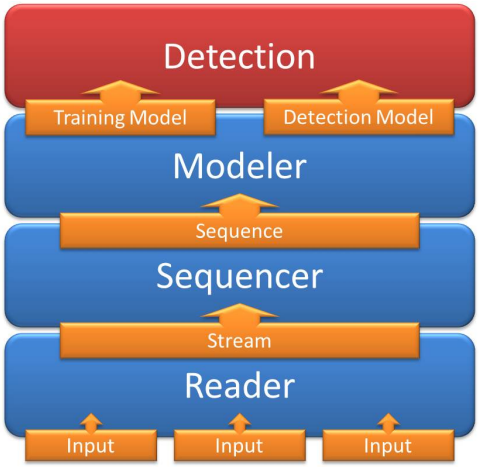

Figure 1: S-IDS general architecture

The *Modeler* is the last layer that performs data manipulation. This layer is in charge of reading temporal sequences coming from the Sequencer and building models that represent how the system behaves over time. A refined modeling step is necessary for two reasons. First, it is unfeasible to store and analyze large amounts of long sequences of events without exhausting computational resources. An S-IDS could be interested in long-time behavior of a specific device and capture information for days or weeks. Therefore, a Modeler must act as a data compressor. Second, the detection mechanisms will focus on a set of features contained and hidden within the sequences. For this reason, the Modeler is used to emphasize such features and offer a simple interface to access valuable information. We will later show what mechanisms a modeler will use for this task.

The *Detection* layer describes all detection algorithms used with the models. With respect to the underlying layers it does not completely abstract from Modeler's output. In fact, the implementation of the detection mechanisms is usually linked to the specific data structures of the model and their set of properties. The Detection layer uses a set of "training models" as reference point (either of normal or abnormal behavior) and analyzes the differences of the "detection models". Furthermore, the Detection layer is in charge of arising alerts to users when "detection models" show malicious patterns.

In what follows we discuss possible implementations of an S-IDS. We use two Reader instances to gather information from network messages and log files. Then, we show three possible different Sequencer instances to analyze ICS protocols, host log entries, and the behavior of several process variables. We use discrete-time Markov chains to model sequences. Finally, we implement a detection mechanisms that computes weighted-distance statistics among Markov chains based on the probability of their transitions.

5. SEQUENCING

Detecting sequence attacks relies on effectively sequencing and modeling devices' operations and behaviors. In this section we describe how to transform a trace of messages, commands, log entries, or similar into ordered lists of events. The *event* represents the abstraction we use to arrange in-

formation in a chronological order. As described in Sec. 4 we show three different use cases: network communications, host log entries, and process variable values.

5.1 Network communication

A unique definition of "sequences of events" in the context of network communications cannot describe in details the variety of protocols, communication types (e.g., synchronous vs. asynchronous) and communication patterns (e.g., pushing vs. polling) being used in ICS. For this reason a specific definition of each kind of communication is needed. We extensively treat this topic in [3] and we derive three different definitions for Modbus, MMS, and IEC104. For the purpose of this paper, we report here the definition of a Modbus sequence:

Definition 1. A Modbus sequence is a time-ordered list of events $\{e_{t_n}\}$ where e is a 3-tuple <ID, Code, Data> derived from a sequence of message pairs $(m^{Req}{}_{t_n}, m^{Res}{}_{t>t_n})$.

ID is the "Transaction Identifier", *Code* indicates the type of operation performed, and *Data* represents the information carried by Modbus requests and responses (its semantic depends on the "Function code" and can represent addresses, number of coils or registries, numerical values, etc.). In case requests or responses do not have a match (e.g., unanswered requests) the pair will reduce to a single message.

In Section 8 we test an S-IDS instance on a Modbus communication network.

5.2 Log Files

A log file intrinsically defines a sequence of operations or events within a system or a network. For this reason, the definition of a "sequence of events" simply involves the selection of a proper subset of entries within the log.

Log files used in ICS are likely to record a large set of heterogeneous events that occur during ICS operations. Depending on their nature, they can store information about a specific system or store data related to communications among devices. Events can be notifications, commands, alerts, errors, etc.. Moreover, there are different possible triggers of such events (e.g., human operators, time-scheduled operations, etc.).

For the purpose of modeling an ICS' "normal" behavior, a sequence defined on a log file could avoid alerts, errors and any other event that identifies a problem. However, we are aware of situations in which operators deal with recurrences of warnings caused by correct operations. For this reason we argue that only human activities such as logins and logouts should not be part of the model due to their high time variability.

Therefore,

Definition 2. A logfile sequence is a time-ordered list of events $\{e_{t_n}\}$ where e is a n-tuple representing the attributes of a log entry that identify an operation performed by a device.

When the structure of the log is known, the previous definition can be further refined by filtering specific fields (e.g., redundant information).

5.3 Process Variables

A "sequence of events" related to a process variable describes how its value changes over time (as shown in [11]).

For this reason,

Definition 3. A process variable sequence is a time-ordered list of events $\{e_{t_n}\}$ where e is the value of the process variable at time t.

Usually, ICSs deal with thousands of variables that represent physical or control parameters. Not all of them are suitable for a sequence analysis and the benefit given by modeling such data depends on its semantic meaning. As an example, modeling a quantity such as a temperature or a pressure can give useful insights on the behavior of a system component and its physical boundaries. On the other hand, bitwise variable modeling can result in a random sequence of elements. Finally, two or more process variables can be logically linked together. If this link is a priori known, event e can be a n-tuple of values.

6. MODELING

Once we have established the event sequences, we can create a model of our system using the chosen modeling approach. In this work we use discrete-time Markov chains (DTMC).

As shown in [3], modeling ICSs with DTMCs has some advantages with respect to using other modeling techniques such as n-grams or Deterministic Finite Automata (DFA) and fits our need for a robust and probabilistic model.

In case of discrete-time Markov chains (DTMC), the modeling process clusters in a model's state sequence events that share the same semantic meaning. In the "network communication" case this concept can refer to the involved commands or carried data (e.g., a DTMC state that gathers together all the "write" commands that change a specific variable). In the "log file" case it is possible to cluster sequences of events that share the same identification code or deal with the same component (e.g., read notifications of a device status). Finally, in the "process variable" case, the DTMC states can cluster values belonging to a specific interval (e.g., temperature values discretized to integer scale).

The DTMC transition between two states A and B indicates that, in the sequence under analysis, at least an event belonging to B comes just after an event belonging to A. The construction of a DTMC is independent from the application protocol and it is completely automated. For every event of a sequence, the modeling algorithm either adds it to an existing state ("updateS" function), or creates a new one in case the event does not match the attributes of any other state in the model ("addS" function). Moreover, in each step, the algorithm adds or updates a transition function that links the previous visited state to the current one ("addT" and "updateT" functions respectively). Algorithm 1 shows the DTMC modeling process.

Every state S is defined by a 5-tuple <Data, Type, #Events, FTS, LTS> consisting of:

- **Data**: information that univocally identifies S and describes the portion of information that its events share with each other
- **Type**: attribute that indicates if S represents elements that include: request/response pairs, just requests or just responses (in case no matches are found)
- **#Elements**: number of elements found in the sequence that belong to S
- **First Time Seen (FTS)**: timestamp of the first element annexed to S

Algorithm 1: ModelingDTMC(*sequence*)

```
forall the e_{t_n} ∈ sequence do
    State_{DTMC} ← extract_attributes(e_{t_n});
    if State_{DTMC} ∈ DTMC then
        updateS(State_{DTMC});
    else
        addS(State_{DTMC}, DTMC);
    if Transition_{previousState,State_{DTMC}} ∈ DTMC then
        updateT(Transition_{previousState,State_{DTMC}});
    else
        addT(Transition_{previousState,State_{DTMC}},
        DTMC);
    previousState ← State_{DTMC};
```

- **Last Time Seen (LTS)**: timestamp of the last element annexed to S

Every transition function δ from a source state (Src) to a destination state (Dst) is defined by a 6-tuple <#Probability, Jumps, FJ, LJ, ATE, σATE> consisting of:

- **Probability**: the ratio between the number of jumps from Src to Dst and the total number of jumps from Src to any other state of the DTMC
- **#Jumps**: number of jumps from Src to Dst found in the sequence
- **First Jump (FJ)**: first occurrence of this transition in the sequence
- **Last Jump (LJ)**: last occurrence of this transition in the sequence
- **Average Time Elapsed (ATE)**: average time between two consequent states of Src and Dst
- **Standard Deviation on Time Elapsed (σATE)**: standard deviation calculated over all occurrences of this transition

To illustrate how the DTMC modeling process works, let us consider the following sample sequence built on the "Modbus network communication" use case:

1. **Modbus "Request/Response" element**: Tran. ID = "134", Unit ID = "1", Function Code = "1" (Read Coils), Data = "Starting address = 0, Quantity of coils = 1", Timestamp = "1st Sep. 2013 10:15:32.032"
2. **Modbus "Request/Response" element**: Tran. ID = "135", Unit ID = "1", Function Code = "3" (Read Holding Registers), Data = "Starting address = 100, Quantity of registers = 10", Timestamp = "1st Sep. 2013 10:15:33.126"
3. **Modbus "Request/Response" element**: Tran. ID = "136", Unit ID = "1", Function Code = "1" (Read Coils), Data = "Starting address = 0, Quantity of coils = 1", Timestamp = "1st Sep. 2013 10:15:34.983"
4. **Modbus "Request/Response" element**: Tran. ID = "137", Unit ID = "1", Function Code = "3" (Read Holding Registers), Data = "Starting address = 100, Quantity of registers = 10", Timestamp = "1st Sep. 2013 10:15:35.033"

As shown in Figure 2, the first event of the sequence creates a DTMC state of type "Request/Response" representing Modbus messages used to read 1 coil from address 0 (State A). The second element of the sequence creates another state in the model, which has the same "Type" attribute but represents Modbus messages used to read 10 registers from address 100 (State B). At this point, the model has also a transition between the two states that describes the sequentiality between the two elements of the sequence (Transition 1) as shown in the figure. The third element of the sequence has the same attributes as the first element, and thus, will increase the attribute "Number of elements" of the first DTMC state. However, a new transition connects back State B to State A to show the new relationship observed in the sequence. Finally, the forth element of the sequence is part of State B since it has the same attributes as the second element of the sequence. As the model already has a transition linking the two states, the algorithm only increases the attribute "Number of jumps" of the transition that connects State A to State B and the statistical attributes get updated. Figure 3 shows the final DTMC.

In the next section we now implement a detection mechanism that measures how much ICS parameters change over time with respect to the DTMC we just built.

7. DETECTION MECHANISM

During the learning phase, a sequence-aware NIDS uses the techniques shown in Section 6 to build a model of regular behavior. Our assumption is that no malicious activity has yet been performed until this point, i.e., the learning input is free of malicious anomalies. During the detection phase we now search for unknown or uncommon patterns that are effects of semantic or sequence attacks carried by intruders.

7.1 Targeted Anomalies

Targeted anomalies can be of three types:

- *Unknown State*: relates to a Markov chain state created during the detection phase that does not appear in the model. This anomaly can be the effect of a semantic attack when an event has known attributes (attributes shown by different Markov chain states modeled during the training phase) but attributes do not match correct or plausible values (e.g. out-of-bound physical measurements).

- *Unknown Transition*: relates to a Markov chain transition that has not appeared in the training phase. This anomaly can be the effect of an order-based sequence attack when such a transition links two or more events whose effects together cause damage to either the control or the physical process (e.g., a specific sequence of control operations as showed in [8]).

- *Unknown Probability*: relates to known Markov chain states and transitions whose related probabilities differ (by at least a value of θ) with respect to the one detected during the training phase. This anomaly can be the effect of a time-based sequence attack when the repetition (or the lack) of occurrences of two or more connected events cause damage should cause damage to the ICS (e.g., the repetition of two conflicting events as shown in [24]).

While identifying the aforementioned anomalies, the detection algorithm has to avoid the following false positives:

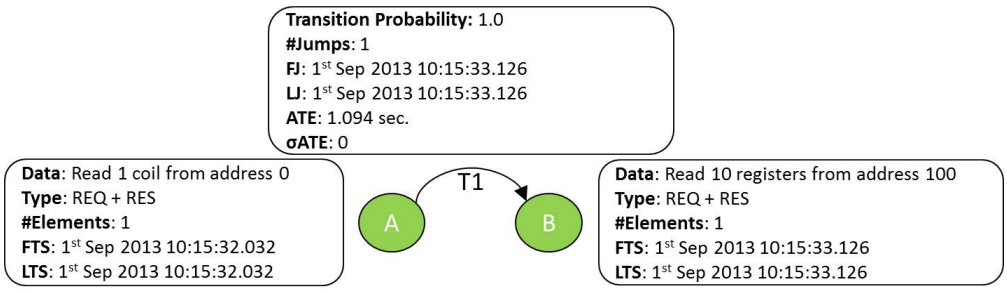

Figure 2: DFA modeling algorithm (phase 1)

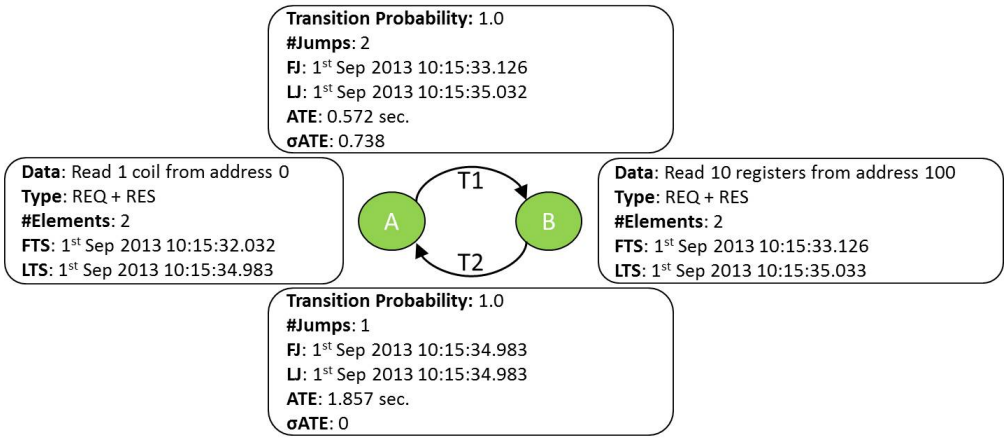

Figure 3: DFA modeling algorithm (phase 2)

- *Unknown but Correct State*: relates to an "Unknown State" anomaly caused by an event that is not dangerous for the system under control. This can occur in case of uncommon user operations or changes either in the control or physical parameters. A S-IDS should be able to identify such a situation as anomalous but label it as "harmless".

- *Unknown but Correct Transition*: relates to an "Unknown Transition" anomaly caused by a new link between two events that does not interfere with the correct functioning of system processes. This can occur in case of event delays or user interferences within control or physical processes. As in the previous case, an S-IDS should be able to identify anomaly's inoffensive nature.

We argue that it is unlikely to have many false positives related to "Unknown Probability" anomalies. This mainly comes from the robustness of such a parameter. Substantial changes in the probability values of a Markov chain mean a considerable modification in the way events are correlated within the system under control. Actions of this kind cannot be frequent. For this reason a S-IDS should always notify "Unknown Probability" anomalies.

Finally, it is worth noting that aforementioned false positives are mainly related to the training time. The more the S-IDS is able to train (the more data it acquires on correct events and patterns) the more it will be able to distinguish a malicious activity.

7.2 Detection algorithm

In the detection phase we evaluate differences between trained DTMCs and DTMCs built up during detection. The detection mechanism flags as "anomalous" a DTMC state created in the detection phase that does not match with any state of a DTMC created in training phase. The same check is performed by looking for new transitions (between known states). Finally, given a known DTMC state (a state included in one of the training models), the detection algorithm looks at its transition set and computes the difference between the probability values measured in the learning phase and the new ones. In the same way, given a known DTMC transition, the detection algorithm compute the difference between the probability values measured in the learning phase and the new ones. This two distances are defined as follows:

$$d_S = \sum_{t \in T} \frac{|p_{t_{detection}} - p_{t_{learning}}|}{2} \quad (1)$$

$$d_T = |p_{t_{detection}} - p_{t_{learning}}| \quad (2)$$

where:

- T is the set of transitions belonging to state S
- $p_{t_{detection}}$ is the probability value of transition t on detection phase
- $p_{t_{learning}}$ is the probability value of transition t on learning phase if such transition exists (0 otherwise)

The result of both Equations 1 and 2 is always in the range of $[0, 1]$.

The detection procedure based on DTMC state distances defines a threshold θ. If the result of Equations 1 or 2 exceeds θ, the detection mechanism triggers an alert to the user showing the involved DTMC state and its semantic meaning. The proper definition of the threshold value strongly impacts the performance of a "sequence-aware" NIDS. If θ is too high the accuracy of the NIDS increases (fewer false positives) but its comprehensiveness decreases (more false negatives). On the other hand, if θ is too low the resulting high number of false positives is likely to make the NIDS ineffective. The threshold value also depends on the nature of the environment in which the NIDS is working. Control systems that show higher variability in the communication patterns will need higher thresholds as well. Our later experiments will discuss selection of an appropriate θ.

Finally, it is worth noting that the detection algorithm can work both offline (e.g., on an early captured network trace) and online (i.e., running on live systems). Offline detection takes place only at the end of the traffic analysis. Once the NIDS goes through all the network frames it builds the DTMC and compute all the distances. Online detection requires instead a further refinement. In fact, the DTMC is progressively populated as the network frames reach NIDS sensors. At the beginning the number of jumps and transition in the DTMC is necessarily low. This likely causes the probability values to diverge from those observed during the learning phase. For this reason, online detection needs to wait a reasonable amount of time τ such that the number of network frames observed in the detection phase is comparable with the number of frames captured and used in the learning phase.

7.3 Enhancing detection

To improve the detection algorithm we decide to leverage information of the infrastructure and the physical process beyond.

7.3.1 Event Importance

Not all the events have the same importance within a system. Different messages, commands, and values have different impact on controlled systems. As a consequence, some messages, commands, and values are more likely to be used within semantic attacks. If this information is known, it is possible to enhance detection mechanisms by focusing the analysis on just the "important" events. A S-IDS that leverages this information will react differently if an anomaly relates to an "important" or an "unimportant" event and will trigger an alert on the second only if the change is very high. This mechanism, if correctly implemented, can significantly reduce the number of false positives.

Evaluating the importance of an event requires a good knowledge of the system under control. For this reason, the most reliable way to set importance values is to leave the decision to a human operator. However, the number of choices that a human operator can take is probably very limited with respect to the amount of events a S-IDS will process. Therefore, we propose two different (yet inter-operable) ways to apply and use importance values within models.

The first approach relies on making some a priori assumptions on the importance of some events. In the network communication example, we can roughly divide ICS messages into two categories: "monitoring" messages and "control" messages. The former category groups together messages that are used to gather information about the status of a system. The second one includes all the messages used to change this status and directly take action on the system. From the attacker's perspective we argue that "command" messages are more important than "monitoring" ones.

This is due to our threat scenario. As described in Section 1, the goal of the attacker is to damage the infrastructure by subverting the control process. To do that, the attacker needs to exploit messages that interfere with the control process and thus, change its status. For this reason, in this scenario, we could tag DTMC states that represent "write" or "execute" operations as "important", while tagging all the others as "unimportant".

The second approach relies on the presence of some events already tagged as "important" and on making assumptions on the others based on their proximity with such "important" events. In the case of DTMC model, the importance of some states is transmitted and distributed to the others according to the probability transitions.

As example, we define an algorithm that measures the degree of importance of every state of the DTMC from a given subset of "important" DTMC states (Algorithms 2 and 3[2]).

Algorithm 2: calculateImportance()

```
forall the state ∈ DTMC do
    visitedStates ← state;
    p = 1;
    calculateImportance(state, p, visitedStates);
```

Algorithm 3: calculateImportance(*state*, *p*, *visitedStates*)

```
probability = 0;
forall the s ∈ neighbors(state) do
    if s ∈ importantStates then
        probability += p * probability(state,s);
    else if s ∈ visitedStates then
        visitedStates ← s;
        probability += calculateImportance(s,
                                probability, visitedStates)
```

If the concept of "importance" is used, Formula 1 and 2 modify as follows:

$$d = w_s \cdot \sum_{t \in T} w_{s_t} \cdot \frac{|p_{t_{detection}} - p_{t_{learning}}|}{2} \tag{3}$$

$$d_T = w_{source} \cdot |p_{t_{detection}} - p_{t_{learning}}| \tag{4}$$

where:

- $p_{t_{detection}}$ and $p_{t_{learning}}$ are still the probability values of transition t on detection and learning phases respectively

[2]In Algorithm 3 the variable *probability* represents the sum of the probabilities to arrive to an "important" state starting from *state*.

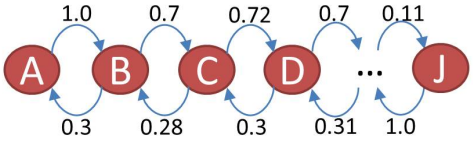

Figure 4: Training model example

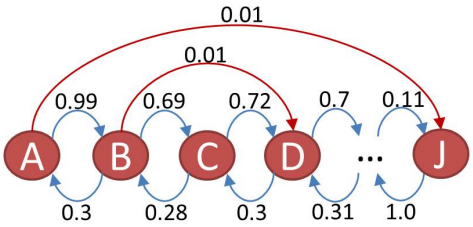

Figure 5: Detection model example

- w_s is the "importance" of the state under analysis
- w_{s_t} is the "importance" of the neighbor state linked through transition t
- w_{source} is the "importance" of the state to which t belongs

The result of Formula (1) is also in the range of $[0, 1]$ if the maximum value of importance is 1. E.g., we can assign 1 to the "important" states and 0 to the others and run Algorithms 2 and 3 with the assurance that all resulted importance values will still be defined in $[0, 1]$.

7.3.2 *Event semantics*

In this work we propose algorithms that use semantics to model events. However, when it comes to alerting anomalies, the detection mechanism just verifies if a state or a transition are included in a training model and it computes weighted-distance between known DTMC states.

A further semantic analysis of an anomalous state (or transition) has a twofold value. First it can force the S-IDS to revise the result and to drop the alarm (e.g., if the anomaly does not fit the DTMC but fits system's expected behavior at a higher level of abstraction). Second it allows the operator who is reacting to the alarm to better understand the context in which such an alarm has been raised.

Let us suppose we want to model the behavior of a variable that represents a temperature. DTMC's states can be defined as intervals of n degrees (e.g., State "A" represents $[0°, 10°]$, State "B" represents $[10°, 20°]$, etc.). The value of the variable ranges between $0°$ and $100°$. Figure 4 shows a possible DTMC built on this hypothesis. Furthermore, let us suppose that during the detection phase two anomalies are present leading to two new transitions in a detection model (Figure 5).

According to the detection algorithm, the two anomalies have the same detection score. However a further semantic analysis of such transitions can help the system to rank the two anomalies and to give a better suggestion to the operator. The two transitions are, in fact, very different in meaning as one concerns a temperature increase of $\sim 20°$ while the other represents an increase of $\sim 100°$. While the first anomaly can be explained by an imprecise sensor or a delay in the update of the variable's value, the same considerations do not stand for the other. A comparative analysis on the transitions would easily show that the average variation in temperature is at most $\sim 10°$. This would make the first anomaly to be very close to a "valid" behavior while emphasizing the distance between the second anomaly and the rest of the known transitions. An S-IDS could then decide to ignore the anomaly or to forward such information to the operator.

A general application of such an approach is limited due to the heterogeneity of environments and events an S-IDS has to face. However, there are situations that allow the presence of a further level of analysis involving the semantic meanings of specific events. We consider the addition of "semantic distance computation algorithms" (both for states and transitions in the case of a discrete-time Markov chain) as a valuable research direction to improve the effectiveness of a S-IDS.

7.3.3 *Transition timing*

An effective detection on sequences of events strongly depends on transitions' time variability. The more events follow standard time patterns the easier is to spot anomalies. Attributes ATE and σATE provide information on how much a specific transition changes over time and should be used as a reference point to decide if performing detection on the related events. Transitions with low time variability relate to the automated behaviors of control processes. On the other hand, transitions with high time variability can be the effect of human behaviors and should be filter out completely during the detection phase. In [3] we show the effects that human behavior causes on DTMC models of ICS communications.

8. EVALUATION

As introduced in Section 1, we test our approach on data coming from real industrial deployments. What follows shows results gathered by our S-IDS from a water treatment and purification facility that uses Modbus communication. The purpose of this test is to first evaluate whether our detection approach may generate too many false positive alerts. Afterwards we introduce some artificial example attacks into the trace to show the effectiveness against attacks.

8.1 Sequencing and Modeling

Over the four hours of training, the infrastructure shows 20 different Modbus connections: 9 PLC-to-RTU, 3 PLC-to-PLC, and 6 PLC-to-SCADA Server, 1 SCADA Server-to-SCADA Server and 1 HMI-to-SCADA Server. Figure 6 represents the result from our modeling approach applied to a communication involving a PLC and the SCADA server.

Most of the Modbus connections involve just one or two Modbus requests and responses sent periodically (e.g., once every second). Connections between PLCs and SCADA Server have instead higher variety of messages. Not all the observed transitions have the same probability. For example, Figure 6 shows that most of the transitions have low probabilities (thiner lines) while a small set of them have a probability greater than 0.9 and form a clear path through all the states of the model (thicker lines). The sequence of events given by transitions with high probability breaks occasionally due to random delays or human interventions.

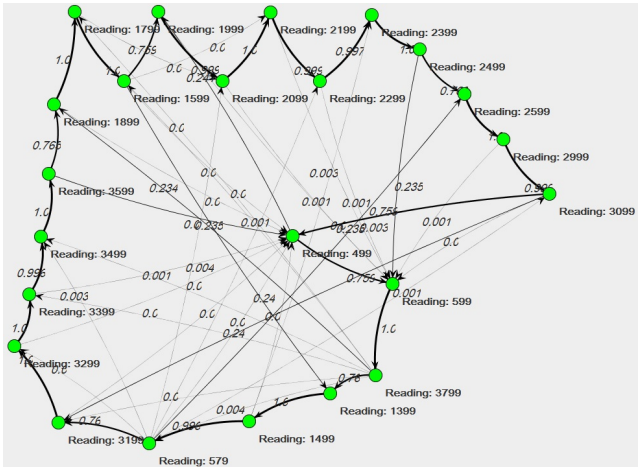

Figure 6: Modbus communication model between PLC "A" and the SCADA Server

8.2 Detection Mechanism

We run our detection algorithm (Equations 1 and 2) against 1 day of traffic captured on the same network. We minimize the threshold θ to 0.1 to alert any significant change within the transition probability sets. No malicious traffic is included in the dataset, therefore this test verifies detection resilience against false positives. The S-IDS raises 211 alerts (Table 1) with a rate of 1 alert every ∼40 minutes. 96% of the false positives are generated by a new transition due to a network delay on a "read" operation within one of the aforementioned connections between PLCs and SCADA server. Other anomalies are: six new states (write messages that were not observed in the training phase), two anomalous states and an anomalous transition (belonging to one of the previous anomalous states). The two anomalous states relates to a read and a write operation. The read event is alerted because of small changes within its transition set (transition probabilities changed within a maximum of d_t = 0.05). Instead, the write operation is alerted because of the detection score of the anomalous transition (d_t = 0.2). Thanks to the operators we observe that this anomaly is due to a change in the status of a pump. Such a change was already observed in the training. However, the low number of events in that specific Modbus communication caused a probability value to increase substantially.

	Unknown State	Unknown Transition	Unknown Probability
Alerts	6	202	3

Table 1: Alert distributions with non-malicious traffic

To prove the effectiveness of the S-IDS we inject two sequence attacks into the traffic. Discussing with the operators, we decide to perform the following two attacks: 1) we invert two write messages concerning the control of two different pumps and 2) we trigger several start and stop commands on the same pump. The choice of these two attacks follows the two threat scenarios presented in Sec. 2.2. Due to the criticality of the water infrastructure we were not allowed to test the attacks on the real network. For this reason we replay the same 24 hours of real traffic plus the attacks to the intrusion detection system. Also in this case the S-IDS raises the previous 211 false positives for the reasons discussed above. In addition, the S-IDS raises also eight correct alerts detecting both the attacks. The first attack causes the presence of two new transitions (one following the other on the targeted write events). Figure 7 shows the first attack modeled in the DTMC. The second attack causes two anomalous states, a new transition and three anomalous transitions with a substantial change in the probability values (transition linking the "start" event to the "stop" one increases from ∼0.001 to ∼0.99). Figure 8 shows the second attack and its effect on the DTMC. It is worth noting that there is a difference between the new transitions generated by the two sequence attacks. In the first case, transitions have small probability while in the other case the new transition has a high probability. This is due to the differences between the attack scenarios. In fact, the first scenario relies on subverting a specific operation known to be potentially dangerous while the other scenario relies on repeating the same operation multiple times.

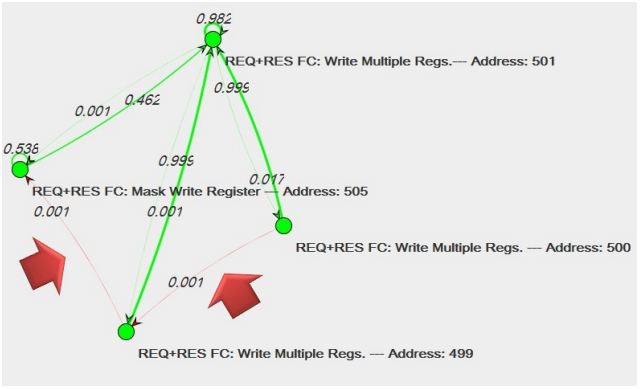

Figure 7: Modbus communication model between PLC "B" and the SCADA Server

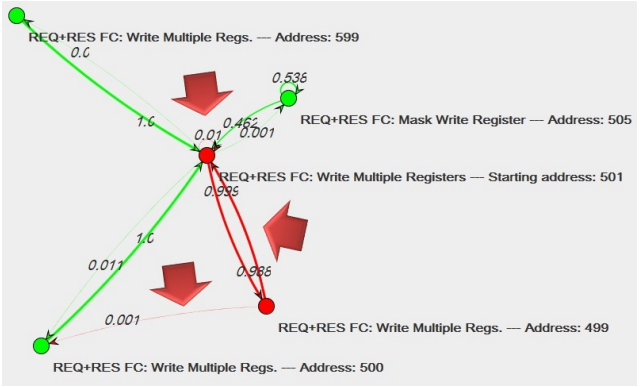

Figure 8: Modbus communication model between PLC "C" and the SCADA Server

8.3 Enhancing Detection

To reduce the rate of false positives we use one of the enhancing methods proposed in Section 7.3. We argued that an attacker is likely to target communications that involve

commands or write operations. Therefore, we define as "important" any command and write operation observed in the communications and assign them a "value of importance" equal to 1. Furthermore, we run Algorithms 2 and 3 to compute the "value of importance" of any other event.

With this new setup we repeat the two tests discussed above. Within the 24 hours of clean traffic the S-IDS raises just 9 false positives instead of 211, with a rate of 1 alert every two hours and a half. This result definitely improves the usability of the intrusion detection system. However, we are aware that, from a practical point of view, such a rate can still be difficult to handle for an operator. Likewise, when adding the malicious traffic, we get 17 alerts. The eight alerts identified before as effects of the sequence attacks are confirmed with the enhanced detection. It is worth noting that detection values computed for the new anomalies of the first attack were already close to the threshold and, thus, close to be considered non-malicious. Such result is confirmed by the new test only because the involved states are rated as "important" ("value of importance" equal to 1). Any other value smaller than 1 would cause the S-IDS to ignore the two new transitions.

These results show the impact that the "value of importance" has on detection. Mistakes in the assignment of such values would strongly influence the effectiveness of the S-IDS. In Section 7.3 we discussed a general approach to the assignment of "values of importance". However, specific infrastructure can benefit of different techniques. The goal of the test was to prove that such enhancing techniques are as effective as a human (e.g., the operator) is able to provide valuable information about the infrastructure and its communications.

9. CONCLUSIONS & FUTURE WORK

This paper discusses sequence attack scenarios within industrial control systems and presents an approach to the development of a sequence-aware intrusion detection system.

State of the art solutions against sequence attacks still face some limitations. This is due to several reasons such as identifying and analyzing the correct sets of information and modeling them into proper data structures. Moreover, such analysis must be robust against message delays and human operations that can occasionally break the stability of ICS communications.

In this paper, we propose a layered architecture to guide the development of an S-IDS. This architecture encompasses in-depth analyses of ICS data and leads to a comprehensive description of devices' behaviors over time. Furthermore, we propose intrusion detection mechanisms aimed to identify anomalies within models of accepted behavior. Finally, we test our approach using real-world data traces gained from a water treatment and purification facility. We show that our S-IDS is able to correctly identify sequence attack instances as well as keeping the number of false positives low.

In our next steps, we will try to identify an even more comprehensive set of detection mechanisms that can be used in a broader variety of industrial control systems. As differences between such infrastructures can be substantial (e.g., physical processes, control strategies) we need to consider more data from a variety of sources. Furthermore, anomalies in communication patterns, logs, and variables' values do not always relate to attacks but can be generated by a number of other causes (e.g., human intervention, device failures, etc). As shown in [3] temporal and statistical analysis can be used to improve IDS capabilities to recognize the cause of an anomaly.

Finally, it is worth noting that leveraging semantic of ICS communications and parameters is a powerful way to enhance security tools' knowledge of the environment in which they are deployed and, therefore, improve their effectiveness. We will also continue to analyze more semantic attack scenarios and testing S-IDS implementations within further real environments.

10. REFERENCES

[1] R. R. R. Barbosa, R. Sadre, and A. Pras. Flow whitelisting in scada networks. *International journal of critical infrastructure protection*, 6(3):150–158, 2013.

[2] D. Bolzoni, S. Etalle, and P. Hartel. Poseidon: a 2-tier anomaly-based network intrusion detection system. In *Information Assurance, 2006. IWIA 2006. Fourth IEEE International Workshop on*, pages 10–pp.

[3] M. Caselli, E. Zambon, J. Petit, and F. Kargl. Modeling message sequences for intrusion detection in industrial control systems. In *Critical Infrastructure Protection IX*. Springer, 2015. To appear.

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

[5] H. Debar, M. Dacier, and A. Wespi. Towards a taxonomy of intrusion-detection systems. *Elsevier Computer Networks*, 31(8):805–822, 1999.

[6] Equipment, IEC Telecontrol. Systems—part 5-104: Transmission protocols - network access for iec 60870-5-101 using standard transport profiles. *IEC Standard*, 60870, 2006.

[7] N. Falliere, L. O. Murchu, and E. Chien. W32. stuxnet dossier. *White paper, Symantec Corp., Security Response*, 2011.

[8] I. N. Fovino, A. Carcano, T. D. L. Murel, A. Trombetta, and M. Masera. Modbus/DNP3 State-Based Intrusion Detection System. *IEEE Advanced Information Networking and Applications, International Conference on*, 0:729–736, 2010.

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

[10] D. Hadžiosmanović, L. Simionato, D. Bolzoni, E. Zambon, and S. Etalle. N-gram against the machine: On the feasibility of the n-gram network analysis for binary protocols. In *RAID*, pages 354–373. Springer, 2012.

[11] D. Hadžiosmanović, R. Sommer, E. Zambon, and P. H. Hartel. Through the eye of the plc: semantic security monitoring for industrial processes. In *Proceedings of the 30th Annual Computer Security Applications Conference*, pages 126–135. ACM, 2014.

[12] S. Haris, G. M. W. Al-Saadoon, A. P. D. R. Ahmad, and M. Ghani. Anomaly detection of IP header threats. *CSC International Journal of Computer Science and Security*, 4(6):497, 2011.

[13] S. A. Hofmeyr, S. Forrest, and A. Somayaji. Intrusion detection using sequences of system calls. *Journal of computer security*, 6(3):151–180, 1998.

[14] ISO 9506-1:2003. Industrial automation systems – Manufacturing Message Specification – Part 1: Service definition, 2003.

[15] M.-S. Kim, H.-J. Kong, S.-C. Hong, S.-H. Chung, and J. W. Hong. A flow-based method for abnormal network traffic detection. In *Network Operations and Management Symposium, 2004. NOMS 2004. IEEE/IFIP*, volume 1, pages 599–612. IEEE, 2004.

[16] T. Krueger, H. Gascon, N. Krämer, and K. Rieck. Learning stateful models for network honeypots. In *Proceedings of the 5th ACM workshop on Security and artificial intelligence*, pages 37–48. ACM, 2012.

[17] T. Lane, C. E. Brodley, et al. Sequence matching and learning in anomaly detection for computer security. In *AAAI Workshop: AI Approaches to Fraud Detection and Risk Management*, pages 43–49, 1997.

[18] M. V. Mahoney and P. K. Chan. PHAD: Packet header anomaly detection for identifying hostile network traffic. 2001.

[19] G. Mao, J. Zhang, and X. Wu. Intrusion detection based on the short sequence model. In *7th IEEE World Congress on Intelligent Control and Automation*, pages 1449–1454, 2008.

[20] Modbus, IDA. Modbus application protocol specification v1.1a. *North Grafton, Massachusetts (www. modbus. org/ specs. php)*, 2004.

[21] W. Scheirer and M. C. Chuah. Network intrusion detection with semantics-aware capability. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, pages 7–pp. IEEE, 2006.

[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. In *9th ACM conference on Computer and communications security*, pages 265–274, 2002.

[23] K. Stouffer, J. Falco, and K. Scarfone. Guide to industrial control systems (ICS) security. *NIST Special Publication*, 800(82):16–16, 2008.

[24] United States President's Commission on Critical Infrastructure Protection and Marsh, Robert T. *Critical Foundations: Protecting America's Infrastructures: the Report*. The Commission, 1997.

[25] K. Wang and S. J. Stolfo. Anomalous payload-based network intrusion detection. In *Recent Advances in Intrusion Detection*, pages 203–222. Springer, 2004.

[26] C. Warrender, S. Forrest, and B. Pearlmutter. Detecting intrusions using system calls: Alternative data models. In *IEEE Symposium on Security and Privacy*, pages 133–145, 1999.

[27] M.-K. Yoon and G. F. Ciocarlie. Communication pattern monitoring: Improving the utility of anomaly detection for industrial control systems. 2014.