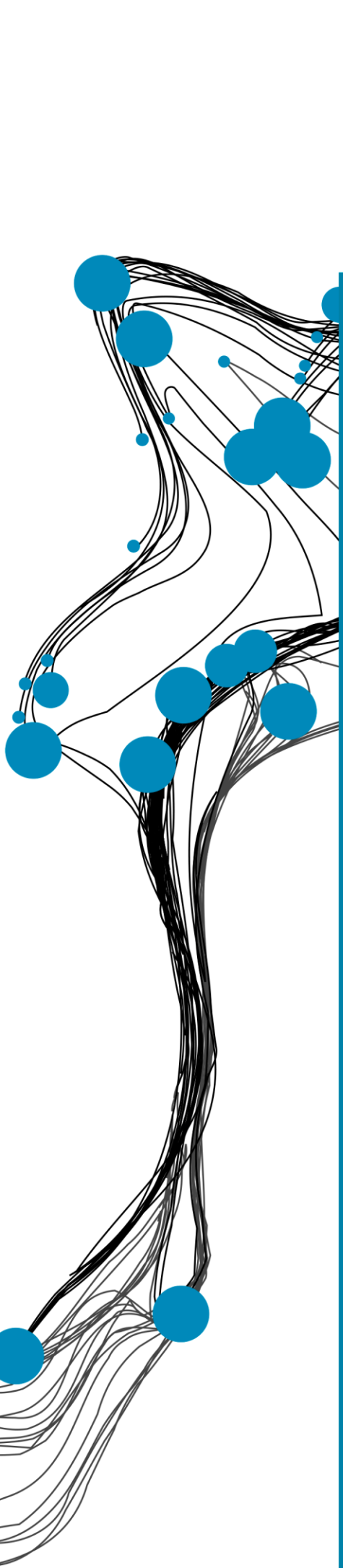

Master thesis

Learning Timed Mealy Machines of the Physical Processes of an Industrial Control System for Anomaly-based Attack Detection

Rayan Brouwer

March, 2020

Supervisors:
Dr. Andreas Peter
Gerrit Kortlever

Faculty of Electrical Engineering, Mathematics and Computer Science

UNIVERSITY OF TWENTE.

Learning Timed Mealy Machines of the Physical Processes of an Industrial Control System for Anomaly-based Attack Detection

Master thesis submitted to University of Twente
in partial fulfilment of the requirements for the degree of

Master of Science

in Computer Science
Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS)

by

Rayan Brouwer

March, 2020

supervisors:
Dr. Andreas Peter
Gerrit Kortlever

exam committee:
Dr. Andreas Peter
Dr. Doina Bucur

UNIVERSITY OF TWENTE.

Deloitte.

March, 2020

ABSTRACT

As Industrial Control Systems (ICS) are turning into automated and highly integrated systems, a closer link between the cyber world and the physical processes is created. Consequently, these critical systems are becoming more prone to cyber attacks. To prevent such systems of becoming unavailable or compromised due to an attack, we propose a method to monitor the physical process and to detect anomalous behaviour. We do this by defining an approach to automatically identify behaviour models of an ICS. Using a machine learning algorithm, state machines are inferred from time series data of sensors and actuators. The normal behaviour of these devices is modelled as Timed Mealy machines, identifying one per subprocess. The results show an efficient way of identifying the models without needing any expert knowledge of an ICS. By using the models as a classifier, the results show a good performance of detecting anomalous behaviour caused by attacks. For testing and validating our approach we use data from the SWaT testbed, i.e. a Secure Water Treatment testbed which is a scaled down representation of a water treatment plant. Out of 36 attack scenarios that were launched on the testbed, our approach detected 28 attacks correctly. The final precision rate shows us that of all the triggered alarms, around 85 percent is relevant. The final attack detection approach is also suitable for other types of industrial control systems.

March, 2020

ACKNOWLEDGEMENTS

I do not think I have all the right words to describe my last year. Of one thing I am sure though: it has been one hell of a roller coaster ride. The process of writing my thesis was accompanied by many difficult moments that made this period quite a journey. However, I had a lot of fun moments too, and I am so grateful for all the things I learned and all the people I met. Because of this, I wanted to take this opportunity to thank everyone who supported me last year.

First of all, I want to thank my supervisor Andreas. You always asked me the right questions about my research, such that I could keep improving my project. You always had the time to help and showed genuine interest in my thesis and subject.

Special thanks to Gerrit, my supervisor at Deloitte. You have been a personal coach for me during the whole period and you always knew the right way to be there for me.

Thanks to all the new friends I made at the Deloitte Cyber team, which I could not have imagined up front. I could not wish for any better place to write my thesis. I had the best time at Deloitte and think this would have been the most fun way to write my master thesis. I already felt like part of the team. Thanks to all my colleagues that took the time to read my thesis and gave me feedback.

And last, but certainly not least, I would like to thank my family and friends, for always being there for me ♡

Rayan Brouwer
March 2020, Amsterdam

March, 2020

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

CIA Confidentiality Integrity Availability

CPS Cyber-Physical System

DCS Distributed Control System

FST Finite State Transducer

HMI Human Machine Interface

ICS Industrial Control System

IDS Intrusion Detection System

MIMO Multiple Input Multiple Output

OT Operational Technology

PDRTA Probabilistic Deterministic Real-Time Automaton

PLC Programmable Logic Controller

RTI+ Real-time Identification from Positive Data

SCADA Supervisory Control and Data Acquisition

SWaT Secure Water Treatment testbed

TMMM Timed MIMO Mealy Machine

March, 2020

1 INTRODUCTION

Industrial control systems (ICS) are experiencing a big transformation since the last decade, also known as the fourth industrial revolution or Industry 4.0 [1]. ICSs are becoming more innovative, automated and highly integrated systems [2]. These systems are creating a closer link between physical processes and the cyber world by using modern computing and communication infrastructures [3]. The term industrial control systems includes control systems that monitor and control industrial processes in different types of sectors and infrastructure such as manufacturing, distribution and transportation [4]. ICSs also control life-critical industries such as power plants or water distribution. Misuse of such a system can lead to serious damage to the environment or can have harmful consequences for human beings, which means that these systems are a high risk when being attacked.

Since Industry 4.0 these critical systems became more prone to cyber attacks, mainly because of the increasing connections with the Internet, but also because nowadays ICSs make more use of commercial software and open architectures to reduce costs and increase the ease of use [5]. ICSs evolved into highly interconnected systems controlling physical processes and consist of a wide variety of equipment with many interdependencies [6]. This leads to increased system complexity and an increasing need for ICS security research.

ICS security differs in many ways from the traditional IT security because of the cyber-physical nature of an ICS. For example, ICSs have many different risks and priorities than IT systems, including the safety risks for human lives and the environment [4]. When considering the CIA triad, the main focus for a classical IT system is mostly the confidentiality and integrity of data while ICSs usually prioritise the availability and integrity of the systems due to the physical processes that are involved here [4].

In the literature ICSs fall into the category of cyber-physical systems (CPS) and often these terms are used interchangeably [6]–[8]. Since the start of the Industry 4.0 trend a lot of research is done on these type of systems. Cyber-physical systems include all systems that combine multiple physical, computing and networking components [9]. Important characteristics of these systems are real-time communication, information processing, remote control and a highly interactive network [9]. Because ICSs have changed from isolated systems to smart and connected cyber-physical systems, the attack surface has become bigger and the number of threats have increased [3].

For instance because nowadays ICSs make more use of Ethernet-based network protocols for communication between all the physical devices, control systems and the human interfaces, some attacks are inherited from IT. Examples can be DOS and DDOS attacks, replay attacks and MITM attacks [6]. These type of cyber-threats do not directly influence the physical processes of an ICS.

In this study the focus is on attacks related to operational technology (OT), e.g. the physical threats, and then specifically the *semantic attacks*. These are defined as attacks that require specific knowledge of the physical systems including the protocols, software, hardware etc., with the main goal to inflict damage on the physical processes. These attacks mostly have a specific goal for a specific target, compared to a DDOS attack whose aim is to disturb a system. Examples of semantic attacks are false data injection attacks, e.g. spoofing data, or sequence attacks which aim to interrupt the normal sequence of events in ICS operations [10].

1.1 ATTACK DETECTION

To prevent or mitigate these attacks from happening, a lot of research has been done on intrusion detection systems (IDS) for CPS, i.e. attack detection [11]. The main objectives of a CPS IDS are collecting data from the industrial process and analysing this data. Different types of analysis can be applied such as pattern matching, data mining or statistical analysis [11]. Mitchell et al. divide the CPS intrusion detection techniques into two main categories which are knowledge-based intrusion detection and behaviour-based intrusion detection [11]. A knowledge-based IDS can be seen as a dictionary, it holds specific patterns of misbehaviour and tries to match these patterns with the real-time data. A big advantage of knowledge-based detection is that it does not trigger many false alarms. However, this type of IDS hardly ever detects new types of attacks. For a behaviour-based IDS, instead of looking for a specific pattern of misbehaviour, here the IDS will trigger an alarm for anything that does not look normal. The main advantage of a behaviour-based IDS is that by looking out for odd behaviour, also unknown attacks can be detected. The disadvantage is that this technique generally produces more false alarms. This behaviour-based technique is also known as *anomaly detection*, as it will try to detect anomalous behaviour.

A variant of behaviour-based IDSs is *behaviour-specification-based detection* and appears to be the most competent technique for intrusion detection in CPSs [11]. This technique will define a *model of the normal behaviour* of a system and an anomaly is detected whenever the monitored behaviour differs from the modelled behaviour. A big advantage of this technique is that it has a very low false-negative rate, meaning a lot of intrusions will be detected correctly. The disadvan-

tage of this method is that it often requires expert knowledge and thus a lot of effort to specify the behavioural models [11].

1.2 PROBLEM STATEMENT

Cyber-physical attacks on industrial control systems can have quite a big impact for human safety or the environment, therefore it is important to be able to detect malicious behaviour in these systems. Although these systems can be complex nowadays - due to the many components that are spread over a possibly large geographical area, the interdependencies and the increasing connectivity - the physical processes controlled by the the systems do not often change that much. The focus in this thesis will be on the physical process, the lowest layer in an ICS. By placing the attack detection on a lower layer, the attack surface that will be covered becomes bigger [6]. More on the different layers of an ICS architecture can be found in Section 2.1.

As was mentioned before, it requires a lot of expert knowledge to specify a precise behaviour model of a physical process that can then be used for an *anomaly-based attack detection* approach. Furthermore, manually creating a behaviour model of a system can become quite difficult with the advanced and complex systems nowadays. And although often is assumed that such models are already created during a system development phase, this is usually not the case [12]. In the literature it is often referred to as the *modelling bottleneck* [13]–[15]. Common problems that emerge from manual modelling, in addition to the complexity and the need for expert knowledge, are (1) internal variables that may not be observable but are needed to define the system's dynamics, (2) it requires a lot of resources, and (3) when systems are updated, the model also needs to be manually updated [13].

Therefore it is useful to be able to learn the behaviours and structure of a system using machine learning which requires less manual effort and system knowledge, and is able to automatically update the model if any changes occur. Learning the behaviour of a system can also be seen as reverse engineering a system. However, there are some pitfalls on learning a model. For example, no attacks can be happening when learning the model of the normal behaviour. Even small disturbances can affect the behavioural model.

A problem that occurs with attack detection is for the operator to handle the amount of alarms that are triggered and to comprehend the detected anomalous behaviour. For this the operator needs to have a good understanding of the processes itself to be able to understand the anomalies. Therefore it would be useful to have an accurate graphical representation of the processes.

1.3 RELATED WORK

Creating a behaviour model of a control system can be beneficial for various types of tasks. In this study the model will be used to detect anomalies in the behaviour of a system, however these models can also be used for model-based system design, to identify faults in a system, or to assure the quality of a system [15]. As creating a model of a system manually can become complicated and time consuming, the automation of establishing these models is quite valuable. System behavioural models can be learned by using machine learning approaches such as support vector machines, neural networks, decision trees or state automata [16].

One of the first related studies on behaviour-based anomaly detection using the SWaT testbed is from Goh et al. [17]. They also used machine learning by modelling the normal behaviour using recurrent neural networks (RNN). They only focused on the first subprocess of the testbed due to time limitations, and included only 10 attack scenarios for evaluating the detection technique. Another related study that also tested anomaly detection on the SWaT testbed is from Inoue et al. [18]. They compare two different kind of unsupervised machine learning methods: deep neural networks (DNN) and support vector machines (SVM). Similar as in this study they first train on the data of the normal operations of the system without any attacks, and then evaluate the the two methods using the four days of testing data. Lin et al. [19] also used machine learning approaches for anomaly detection. They learned a model of the normal operations of the sensors of the SWaT dataset as a timed automata (TA). In addition a Bayesian network (BN) model was learned to discover the dependencies between the sensors and actuators. They called their method TABOR. Combining theses two models resulted in better detection rate compared to the two methods suggested by Inoue et al. However, the fusion of the TA and BN model did cause some false negatives.

The study from Medhat et al. [20] shows a framework for inferring Mealy machines (i.e. a type of automata) from input and output traces. They based the automaton generation on Angluin's L* algorithm, which is an algorithm that is often used as a basis for inferring Mealy machines. They used the sensors as output and actuators as input. They first found the correlation between those devices. They said one output, a sensor, can have multiple inputs, actuators, and thus created for every sensor a separate timed automaton. The purpose of this study was however, not to use the learned behaviour models for anomaly detection.

1.4 RESEARCH QUESTIONS

Derived from the problem statement we defined one main question and multiple subquestions that will be answered in this research.

> *Can we infer a precise model of the physical process of an ICS using time series data in order to detect anomalies in process behaviour without needing expert knowledge?*

The main research question can be divided into two separate subquestions. To answer the first subquestion we want to create precise models of the physical processes without needing expert knowledge that can give us also a clear understanding of the behaviour of the processes. For the the second subquestion we use the models to monitor process behaviour and detect anomalies. We define the questions as follows:

RQ1 Is it feasible to map an ICS subprocess to a single model?

1.1 Is it possible to automatically learn a model of process behaviour of an ICS using time series data?

1.2 Is it possible to represent the complete behaviour of an ICS subprocess into a single model?

RQ2 How can we use the behaviour models to monitor the process behaviour of an ICS?

2.1 To what extent are the models able to detect anomalous behaviour?

2.2 To what extent is the anomaly-based attack detection approach able to detect attacks?

1.5 PROPOSED SOLUTION

Four different phases are defined for the final purpose of creating this attack detection technique for industrial control systems. These phases can be found in Figure 1.

Figure 1. Phases of Detection System.
Based on the architecture of the sequence-aware IDS proposed by Caselli et al. [10].

For testing and validating our approach we use data from the Secure Water Treatment (SWaT) testbed. This water treatment testbed is located in Singapore and is created for research in ICS security [21].

As it is not an easy task to access and perform tests on an actual live ICS, a testbed is proven to be a good alternative as it simulates the real physical processes on a lower scale. The creators of the testbed performed multiple different attacks on the testbed and created a labeled dataset of the monitored behaviour. The dataset can be used to analyse attacks and test detection techniques. Both network traffic as data from the physical devices was collected, however in this study only the latter one is used for creating the behaviour models.

To answer the first subquestion we consider the first three phases. In the first phase of our approach the time-series data from the physical properties of the SWaT testbed is collected. This data includes signals from all the sensors and actuators of the testbed. In the second phase the data is transformed into sequences of events in order to learn the behaviour which will be modelled as *state machines* or state automata. A state machine is an abstract system and represents the different states of a system and how it moves from one state to another. Figure 2 shows a very plain example of a state machine modelling behaviour of a valve. In this example, S1 represents the state where the valve is closed. When receiving the input, i.e. an action, to open the valve, the state machine moves to the next state S2.

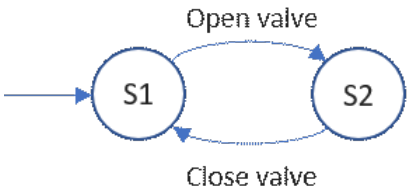

Figure 2. Example of simple state machine.
State S1 represents the valve being closed. When receiving the input to open the valve, it transitions to state S2.

For representational purposes, system or process behaviour is often illustrated as a state diagram, i.e. a graphical representation of a state machine. These state diagrams tend to give a clear description of system behaviour and thus can give, in addition to using the abstract models for anomaly detection, clear insights into the normal and anomalous behaviour. For this third phase - modelling the behaviour as state machines - the tool RTI+ [22] is used. This tool was chosen as it can learn state machines from *positive* time series data, where in this case the positive data is data of the normal behaviour.

The most general state machine, or also known as the *finite state machine*, can move to a next state given an input and the current state. The moving between states is defined by transitions. For representing a control system we want to use a different kind of state machine called the *finite state transducer*. The difference here is that a transducer, in addition to determining the next state, also generates an output. There are two types of finite state transducers which are the

Moore machine and the Mealy machine. Where a Moore machine determines the output based only on the current state a system is in, a Mealy machine determines the output on both the current state and the input value. Figure 3 shows an example of a simple Mealy machine. As can be seen a transition has a pair of symbols (i/o) instead of just one. For example, a system is in a state (S1) where a valve is open and reads an input action "Water level is high". These two values together will not only determine the next state of the system but also an output action, e.g. "Close valve". In this case the input comes from a sensor and the output is the action of an actuator. These characteristics of a Mealy machine match well with the properties of the ICS process behaviour, and in the literature these state machines are considered to be a good fit for modelling the behaviour of real-time reactive systems [23].

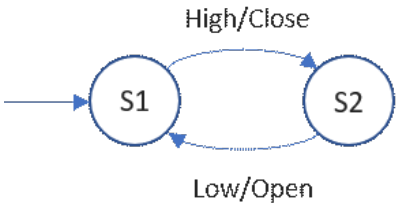

Figure 3. Example of simple Mealy machine.
A transition holds a pair of symbols (i/o) specifying an input and output symbol. In this example the input comes from a sensor measuring if the water level is high or low. The output describes if the valve should be closed or opened.

Where Lin et al. [19] only learn a state machine of a few sensors of the testbed and combining this with a Bayesian network, we propose an even simpler and easier understandable model by including the behaviour of all the devices and grouping them per subprocess. This will result in a variant of the Mealy machine. In the end we will have six behaviour models, one for each subprocess of the SWaT testbed. More on the subprocesses in the SWaT testbed can be found in Section 2.1.1. In addition, as the Mealy machine does not include the concept of time, we will add this by creating *time-based transitions*. As an ICS has real-time requirements, e.g. a valve should probably not stay open for an infinite amount of time, the timing behaviour should be included in the state machines. The Timed Mealy machines will be learned passively from the monitored signals from the sensors and actuators of the SWaT testbed.

To answer our second subquestion we consider the last phase in Figure 1. We want to use the behaviour models to monitor the processes and detect anomalous behaviour. This is done by using the models as a one-class classifier, i.e. the normal behaviour. The attack scenarios that were launched on the testbed are used to validate our approach. We check if this approach is able to detect the anomalous behaviour that was caused by the attacks. As this is a behaviour-based

anomaly detection we can expect many alarms of which some of them are false alarms. Therefore we suggest a prioritisation of the anomalies to, in the end, make it more efficient for the operator to use.

1.6 CONTRIBUTIONS

The contributions of this thesis are as follows. We propose an approach for attack detection in ICSs that (1) creates models of the processes without needing expert knowledge, (2) creates understandable graphical models that precisely model the physical processes which, (3) are able to trigger many correct alarms. And (4) we prioritise the detected anomalies to make the final approach more efficient and manageable for an operator.

We define a variant of the Mealy machine where instead of only an input and an output tape, the machine has more than two tapes, each of them defining the behaviour of a single device. This way multiple signals are aligned to define the behaviour of a subprocess of an ICS. This will result in a Timed Mealy Machine which will give a good and understandable visualisation of a subprocess and in addition can be used to detect anomalous behaviour by using the identified behaviour models as a one-class classifier. The results show that 28 of the 36 attack scenarios are detected, which is more than previous studies using the SWaT testbed. The final precision rate, when combining the six models, shows us that of all the triggered alarms, around 85 percent is relevant. The final attack detection approach should be suitable for many types of industrial control systems.

1.7 THESIS STRUCTURE

This thesis is structured as follows. Chapter 2 provides background information on ICS, the testbed, state machines and the learning process. The methodology of this research is defined in Chapter 3. The behaviour learning of the SWaT testbed can be found in Chapter 4. Chapter 5 shows the results of the anomaly detection. To conclude we discuss the results of the proposed anomaly-based attack detection approach and provide suggestions for future work in Chapter 6.

2 BACKGROUND

This chapter covers background information on the important topics of this research. This includes industrial control systems and the created testbed for ICS security research. In addition this chapter provides the formal definitions for the used models and a small introduction to automata learning.

2.1 ICS SECURITY

The most common types of industrial control systems are supervisory control and data acquisition (SCADA) systems, distributed control systems (DCS) and programmable logic controllers (PLC) [4]. These control systems are for example controlling a physical process such as power generation or distribution, manufacturing or oil and gas refinery. Figure 4 visualises the basic operation of an ICS [4]. An ICS consist of various control loops and human interfaces and makes use of industrial network protocols that can provide real-time control. A control loop employs controllers, sensors and actuators. The sensors receive data from the controlled process, and send this to the controller, the controller being for example a PLC. The controller receives the incoming signals from the sensors, performs programmed instructions, and then sends the output signals to the actuators. A human machine interface (HMI) is used to monitor and operate the controller. The HMI will display all information on the current state of the process and everything that happened before this state.

2.1.1 *A Water Treatment Testbed*

SWaT is a testbed created for ICS security research and is a scaled down representation of a water treatment plant. It can produce 5 gallons/minute of filtered water. The SWaT testbed consists of 6 subprocesses (P1-P6), each controlled by a PLC. The PLCs are networked and communicate with each other constantly to share state information which a subprocess may need from another subprocess. Each PLC receives data from the sensors and actuators. In the SWaT testbed the actuators are for example a pump or a valve. In able to decide whether a pump should be turned ON or OFF, there are sensors that will look at the water level in a tank. Figure 5 shows the different layers of the testbed. A layered communication network is used in SWaT, where the communication between layer 1 and 0 is an Ethernet-based ring network. This ring network communicates data between

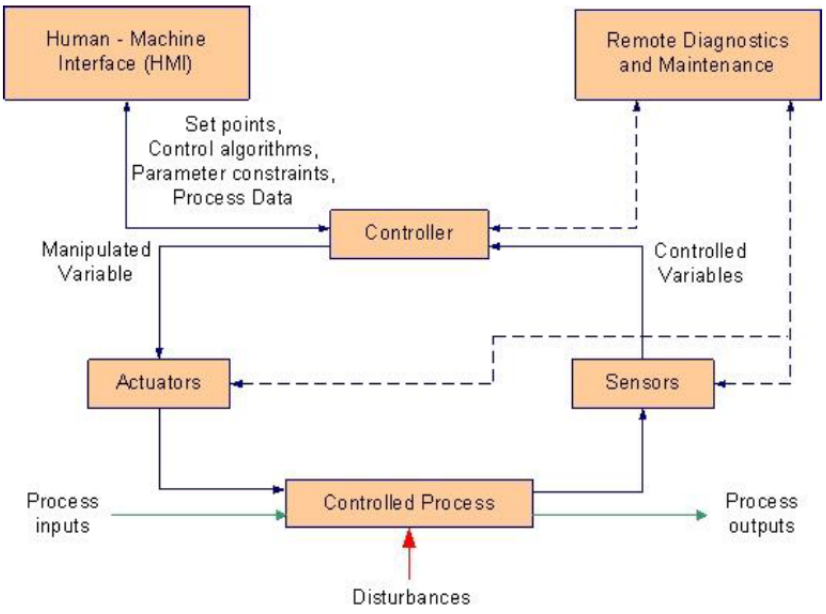

Figure 4. Basic operation of an industrial control system.
An ICS consists of various control loops including a controller (e.g. a PLC), sensors and actuators. Example architecture from NIST [4].

the PLCs and its field devices [24]. The communication between layer 2 and 1 is an Ethernet-based star network where an industrial switch connects the SCADA system and HMI with the six PLCs [21]. There is also a historian that will record the data from the sensors and actuators that is collected by the SCADA system.

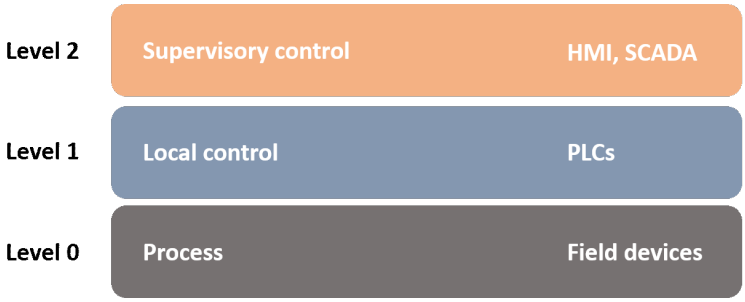

Figure 5. Layers of the SWaT testbed.
The creators of the SWaT testbed based the different layers on a basic ICS network architecture [24].

The six different stages of the water treatment process include the following:

P1 This stage is controlling the inflow of the raw water. Valve can be opened and closed.

P2 Raw water is chlorinated and then pumped into another tank.

P3 The water is filtered using a Ultra Filtration (UF) system.

P4 This stage is a de-chlorination process using ultraviolet lamps to remove any remaining chlorine.

P5 The water will go through a Reverse Osmosis filtration unit. The filtered water is then stored in the permeate tank, and is ready for the distribution.

P6 The last stage is in control of the cleaning of the UF unit.

Figure 6 shows an overview of the six subprocceses of the SWaT testbed [25]. For every subprocess corresponding sensors and actuators are represented in this diagram. These field devices are all given a name such as FIT for a flow meter and P for a pump. The first number that is given indicates to which subprocess the device belongs to. A description of the 51 devices can be found in Table 9 in Appendix A [24].

2.1.2 *Attack Scenarios*

Goh et al. [24] collected data from the SWaT testbed while launching attacks on the testbed. For defining the attacks they used the attack model that is defined by Adepu and Mathur [26]. The attacks were launched through the network between layers 2 and 1 (Fig. 5). Before sending the network packets to the PLCs, Goh et al. manipulated the data from sensors and actuators by hijacking the packets.

They identified different types of attack points such as a physical element (e.g. a sensor or actuator) or a point to access a communication network. Based on the available attack points, Goh et al. defined four different types of attacks [24].

SINGLE STAGE SINGLE POINT (SSSP) This attack only focuses on one point in an ICS.

SINGLE STAGE MULTI POINT (SSMP) This attack focuses on multiple attack points that are present in one stage.

MULTI STAGE SINGLE POINT (MSSP) This attack is a single point attack but performed on multiple stages.

MULTI STAGE MULTI POINT (MSMP) This attack is performed on multiple stages and based on multiple attack points.

In total there were 36 attacks in the dataset that actually affect the physical state of the testbed. As the focus here is on the physical process, only these 36 attacks are included in this study. The dataset contains 23 SSSP, 6 SSMP, 4 MSSP and 3 MSMP attacks. Some attacks are launched in succession without letting the system to fully stabilise before the new attack, in contrast to other attacks where the system

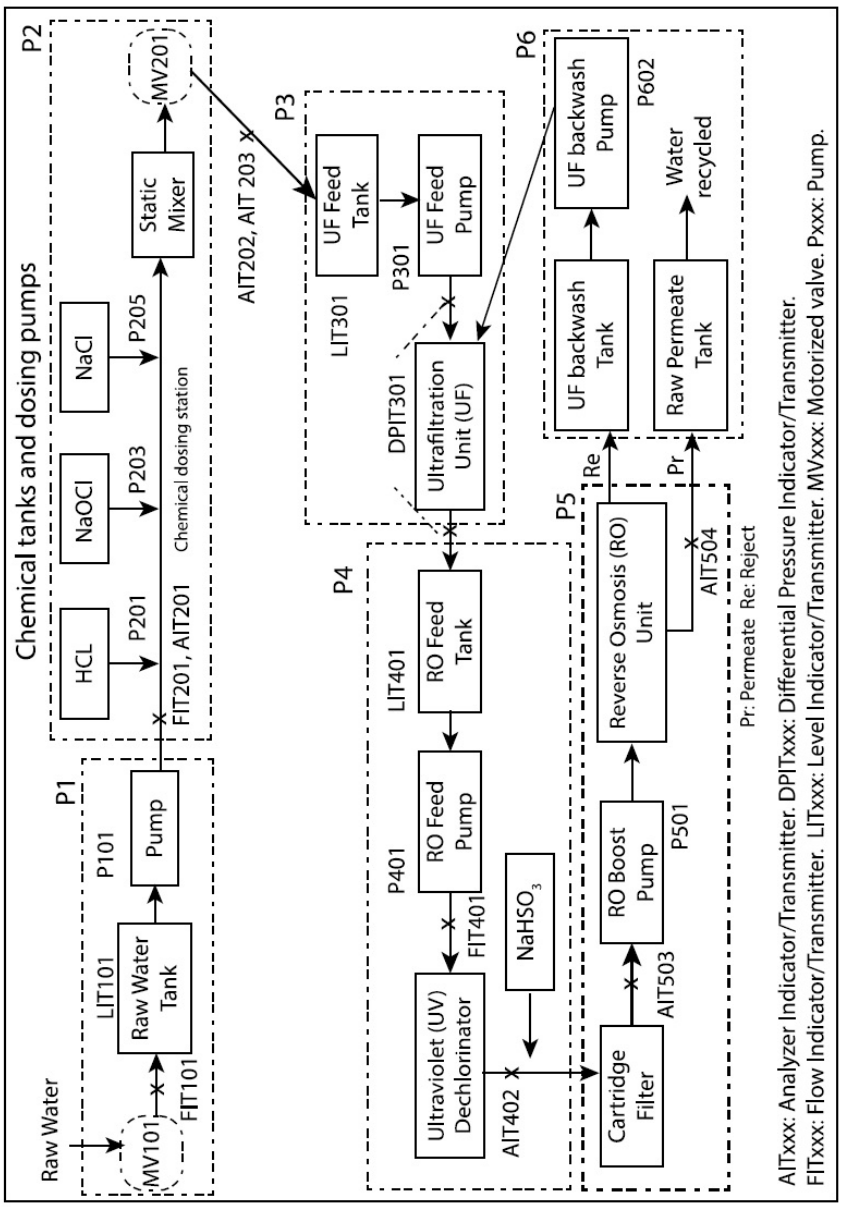

Figure 6. Process overview of the SWaT testbed.
There are six subprocesses (i.e. P1-p6) in the testbed [25]. The field devices are divided by subprocess. The first number indicates to which subprocess it belongs to. Not all devices that are named in Table 9 are in the overview. Some devices in the table act as backup device.

was able to stabilise until normal operations. The difference in the stabilisation phases are due to how advanced the attacks are and what effect they have on the system [24]. A short description of the 36 different attack scenarios can be found in Table 10 in Appendix A. In addition, Table 12 shows some more information on the scenarios including the duration of the attacks, start time and the type.

2.2 STATE MACHINES

State machines are mathematical models which are used in various areas. They are also known as *automata* or finite state machines, as they have a finite number of states. Examples of their use are defining languages, designing protocols and modelling behaviour of numerous applications. A state machine can move from one state to another by receiving inputs. As mentioned before, in this study we use a type of finite state transducers, which specifies a transition with input and output symbols.

2.2.1 *Finite State Transducers*

As elaborated in the previous chapter, in this study behaviour will be modelled as a *finite state transducer (FST)*. Where a normal *finite state machine* only uses an input tape to move from one state to another state, a *finite state transducer* can also generate an output. This can also be seen as a *mapping* between two sets of symbols: the input and the output alphabet. See Definition 2.1.

Definition 2.1 *Finite State Transducer* $A = (Q, \Sigma, \Gamma, \delta, \omega, q_0, F)$ *where* Q *is a finite set of states,* Σ *is the input alphabet,* Γ *is the output alphabet,* $\delta : Q \times \Sigma \to Q$ *is the transition function given a state and symbol to the next state,* $\omega : Q \times (\Sigma \cup \{\epsilon\}) \to \Gamma$ *is the output function where* ϵ *is the empty string,* q_0 *is the start state, and F is a set of final states.*

For every transition in a finite state transducer an input and output symbol is associated. This defines a relation between the input and output alphabet. A transition from one state to another can be defined as: $s \xrightarrow{i/o} s'$. It holds a pair of symbols: i/o. Thus where a normal finite state machine defines a set of accepted strings, a finite state transducer defines the relation between the sets of strings. In this thesis, it will define the *relation between the behavior of the sensors and the behavior of the actuators*.

A *Mealy machine* is a type of finite state transducer, as it holds an input and an output alphabet. Every output is determined by both the current state and the input value. The difference with the Mealy machines is that it does not contain a set of final states. This means that there are no input strings that can be accepted in a final state, instead every transition generates an output. A formal definition of the Mealy machine can be found in Definition 2.2.

Definition 2.2 *Mealy machine* $M = < S, s_0, I, O, \delta, \lambda >$ *where S is a finite nonempty set of states,* s_0 *is the initial state, I is the input alphabet, O is the output alphabet,* $\delta : S \times I \to S$ *is the transition function given a state and input symbol to the next state, and* $\lambda : S \times I \to O$ *is the output of the transition given a state and input symbol.*

2.2.2 *MIMO Mealy Machine*

A more complex variant of the Mealy machine can be a variant with multiple inputs or multiple outputs or both (MIMO). The Mealy machines that will be created of the behaviour of the SWaT testbed will not consist of the mapping between an input tape and an output tape, instead there will be x tapes and the relationship between these x tapes where $x \geqslant 2$. Every x will represent a device (e.g. sensor or actuators) that will be included in the Mealy machine. See Definition 2.3.

Definition 2.3 *MIMO Mealy machine* $MM = < S, s_0, \{D_i\}, \delta, \lambda >$ *where* S *is a nonempty set of states,* s_0 *is the initial state,* $\{D_i\}$ *are the alphabet sets belonging to the devices included in this machine, where* $i \in 0..x$, $\delta : S \times D_0 \rightarrow S$ *is the transition function given a state and the symbol of the first device to the next state, and* $\lambda : S \times D_0 \rightarrow \{D_i\}$ *is the function that outputs the symbols of the other devices given a state and symbol where* $i \in 1..x$

As can be seen in Definition 2.3, δ is similar as in the normal variant of the Mealy machine but instead of using the input alphabet I to determine the next state, the alphabet of the first device is used. Then the output function λ defines the symbols of the other devices according to the first one, i.e. it reads symbol D_0 which then determines the other symbols.

2.2.3 *Timed Mealy Machine*

In order to include the concept of time, transitions will be transformed into time-based transitions. This means adding a time delay guard to the transitions. A transition will become $s \xrightarrow{d_0/\ldots/d_x/t} s'$. This time delay guard defines in what time frame the machine should move to the next state when reading the next symbols.

2.2.4 *Probability*

In addition to time, the learning tool that is used is also able to learn probabilities. This is quite useful for our model of normal behaviour, as the time series data that is used may contain some noise. This leads to modelled behaviour that is not very likely to occur often. Using the probabilities in a transition the model specifies which transitions are most likely to happen.

2.2.5 *Determinism*

A Mealy machine is a *deterministic* FST, meaning that given a state and an input value there is only one transition possible. However, for our Timed MIMO Mealy machine it is possible to have more than one transition given a state and input value if there is a different time frame.

In addition, if we consider D_0 to be the input value, the output is not always deterministic. It might happen that there is a state from which two transitions can be taken, for example with the symbols '1/2/2' and '1/2/1'. In such a case we consider the probabilities of both transitions, as one of the two is presumably noise in the behaviour that was still modelled.

2.3 AUTOMATA LEARNING

Automata learning has been proven to be quite useful in the area of studying unknown behaviour of a system [27]. Two types of learning algorithms can be distinguished here which are passive and active learning. Active learning is also known as query learning and can be defined as a learner (e.g. an algorithm) that will query an oracle (e.g. a system) [23]. This also means that access to the system is required. For a passive learning algorithm this is not necessary, instead it will learn from collected data samples.

In this study an *automaton* (state machine) will be passively learned from a dataset. More specifically, the RTI+ tool that is being used, that implements an automata learning algorithm, will create a probabilistic deterministic real-time automaton (PDRTA). This algorithm is based on the *evidence-based state-merging* algorithm and makes use of a *red-blue framework* [28]. More on this algorithm can be found in the study from Verwer et al. [22], [28]. More on how this tool is used in this study can be found in Section 3.3.

In the following sections the most important steps that are taken in an automata learning algorithm are explained. This includes creating a tree from the data sequences, merging states and splitting transitions.

2.3.1 *Prefix Tree*

In an automata learning algorithm which will learn from data sequences, the algorithm starts with creating an automaton in the shape of a tree [22]. All data sequences are represented in this tree and *sequences with the same prefix are merged together*. These data sequences can be seen as an input string. Every data element of a sequence - or symbol in the input string - is defined as a transition in the tree. An example of a *prefix tree* can be found in Figure 7.

The data strings from which the automaton tree is created are for example the following: **ABC, CD, CDA, ABCDA, BCD, ACDAB**. Two graphical examples of sequences can be found in Figure 8. The figure shows time series data that is represented as a sequence of symbols. Such sequences will also be the input to the RTI+ tool. More information on this tool can be found in Section 3.3.

As time is also included in the learning process, every symbol that is being read and put into the tree comes with a time guard, i.e. (A, 10) , (B, 5), (C, 10) (Fig. 8). In order to learn the time guards for every transition in a later stage, the initial guards for every transition are set to the minimum and maximum values that were observed in the timed data sequences. The minimum observed timeguard is 5 and the maximum value is 15, as symbol D has a time guard of 15. That is why in Figure 7 all transitions get initially time guard [5,15]. A transition can be defined as $< q, q', a, t >$, where a state machine is in state q, and moves to transition q' after time t and after reading symbol a.

Because all the data strings represent the normal behavior of a system (e.g. positive data), the automaton tree does not contain any rejecting states. Also because the behavior is continuous, there are no accepting states.

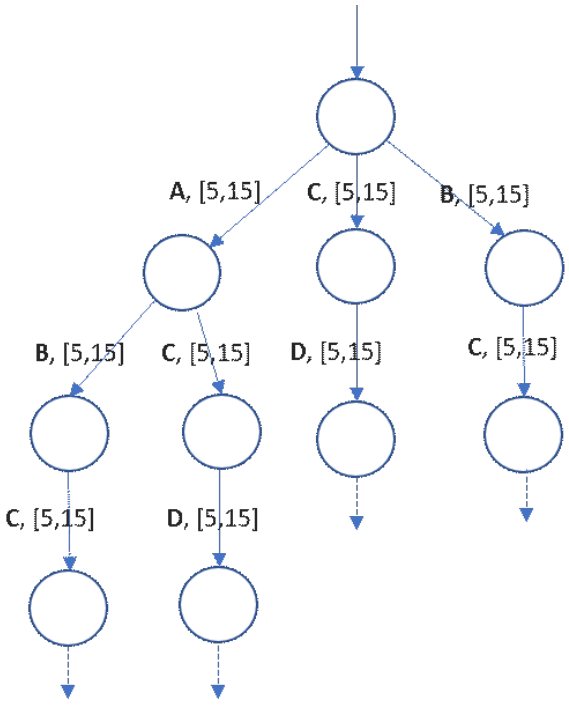

Figure 7. Prefix tree from data sequences.
The first phase of automata learning. All data sequences are put into the automaton in the shape of a tree. A transition includes an input symbol and time frame. As example it shows that sequence ABC and ACD are merged.

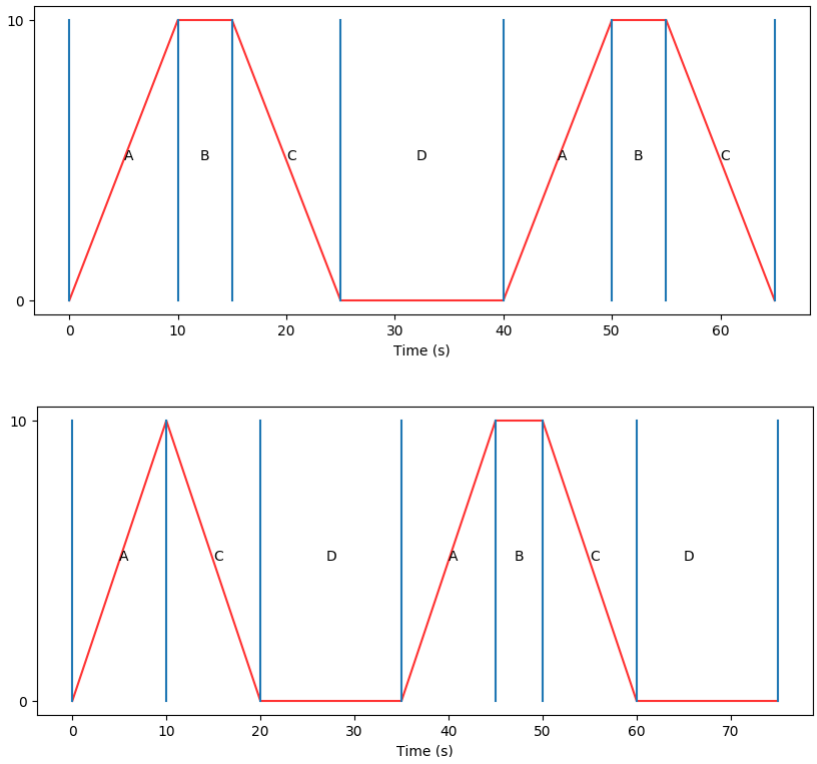

Figure 8. Two examples of time series data sequence.
Input examples for automata learning from time series data. Data will be represented as a sequence of symbols with a timeframe. The second example shows a small difference in the signal which leads to a slightly different sequence of symbols, i.e. ABCDABC and ACDABCD.

2.3.2 *State Merging*

After the automaton tree is created from the data sequences, the actual learning will start. The learning algorithm will try to merge pairs of states to make the state machine as small as possible. In the RTI+ tool the algorithm uses a evidence-based state merging approach to decide whether to merge or not [22].

To decide if two states should be merged, two models will be considered. The automaton before the merging and the automaton after the merging. By applying a likelihood-ratio test the tool can decide if the new model with the merge scores better than the model without the merge. Otherwise the merge will be undone. In this case the likelihood-ratio test is used as the statistical evidence in this evidence-based state merging algorithm.

2.3.3 *Transition Splitting*

Because of the use of a time values in the transitions, a transition $< q, q_1, a, [5, 15] >$ can for example be split into two new transitions:

$< q, q_s, a, [5, 9] >$ and $< q, q'_s, a, [10, 15] >$. The decision to split a transition is made in the same way as the merging process, using a likelihood-ratio test.

3

METHODOLOGY

The different steps that are taken to achieve the objective of this thesis are based on the proposed architecture by Caselli et al. [10]. Figure 9 shows an overview of these steps. In their study, Caselli et al. also propose an approach for a sequence-aware intrusion detection system (S-IDS) and focus on sequence attacks in ICS.

There are four different phases defined in the architecture. It starts with the *reader*, which takes raw data from all the sensors and actuators of the SWaT testbed as an input. This includes the time series data that is used for learning the behaviour model, but also the time series data that is used for testing this model. Finally, for real-time use, this should be data of a certain time frame that will be added continuously, which then can be tested for anomalies.

After collecting all the data, data streams per subprocess will be input into the *sequencer*. For every subprocess the signals of the corresponding devices are joined together as they will define the behaviour of this subprocess. These joined input streams will be transformed

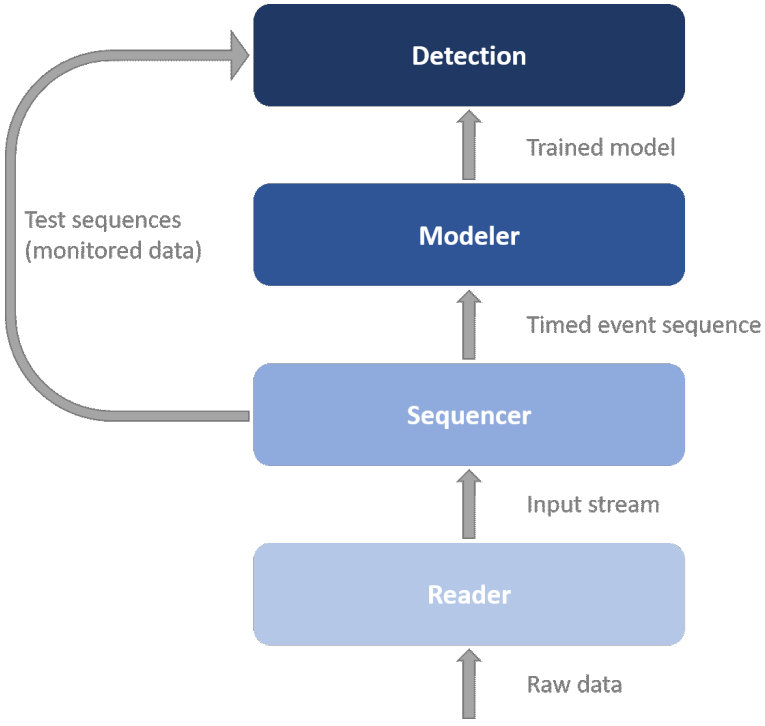

Figure 9. The four phases of this study.
Based on the architecture of the sequence-aware IDS proposed by Caselli et al. [10]. Historical data is used for the modelling. Monitored data or real-time data will go directly from the sequencer phase to the detection phase.

into the right representation such that the next phase can use the data to learn the behavioural model. In able to learn, in our case, timed automata from time series data, the data will thus be transformed into *timed event sequences*.

In the *modeler* phase an existing tool will be used that implements an automata learning algorithm. More on this tool can be found in Section 3.3.

Our *detection* phase differs from Caselli et al. where they explain that the output from the model phase is a trained model and a detection model, we only need a trained model. Instead the test data is coming from the sequencer phase directly, where monitored signals are transformed into timed event sequences. The detection phase is explained in Section 3.4.

3.1 DATASET

To perform this research, collected data from the Secure Water Treatment (SWaT) testbed is used [21]. This Secure Water Treatment system is an operational testbed which is similar to an actual water treatment plant but in a smaller scale. A testbed is a good resemblance of a real-life system in which security measures can be tested in an appropriate way. SWaT is a hybrid system and is designed for doing research on ICS cyber-attacks, detection techniques and ICS security [21].

Goh et al. [24] created a dataset for this purpose including 36 different attack scenarios. There were 11 days of data collection on the SWaT testbed of which there were 6 days of collecting data of the normal behaviour of the system, following with 5 days of behaviour including the launched attacks on the system. The first 6 days will be used as training data, to create the behaviour model of the normal operations of the testbed. The other 5 days, including the attacks, can then be used for testing.

The data was collected from 51 sensors and actuators, hence the dataset holds physical properties of the SWaT testbed which can be used to study the cyber-attacks. The dataset comprises 946,722 samples and 53 attributes. For 11 days, every second one data sample was collected for each of the attributes. All data samples include a timestamp and a label which is "Normal" or "Attack". Table 1 shows an example of the dataset with the physical properties.

Timestamp	FIT101	LIT101	MV101	P101	P102	AIT201	···	Normal/Attack
28/12/2015 10:00:00 AM	2,427057	522,8467	2	2	1	262,0161	···	Normal
28/12/2015 10:00:01 AM	2,446274	522,886	2	2	1	262,0161	···	Normal
28/12/2015 10:00:02 AM	2,489191	522,8467	2	2	1	262,0161	···	Normal
28/12/2015 10:00:03 AM	2,53435	522,9645	2	2	1	262,0161	···	Normal
28/12/2015 10:00:04 AM	2,56926	523,4748	2	2	1	262,0161	···	Normal
28/12/2015 10:00:05 AM	2,609294	523,8673	2	2	1	262,0161	···	Normal

Table 1. Example of SWaT dataset.
In total there are 53 attributes and 946,722 samples. Every sample has a timestamp, the values of the devices at that time, and a label which indicates if this is normal behaviour or if an attack is happening.

3.2 DATA TRANSFORMATION

To be able to use the data as input to the learning algorithm and also as input to the state machine itself for testing, we need to transform the data into sequences. This is done in the *sequencer* phase. The data signals will be made discrete, by splitting them into segments (e.g. events), the segments will be grouped and every group is given a symbolic representation. After the discretisation, *timed event sequences* will be created in order to learn a Timed Mealy machine. Because for every subprocess one Timed Mealy machine will be created, multiple signals need to be combined which should result in a Timed *MIMO* Mealy machine, i.e. a Mealy machine with multiple inputs and multiple outputs. This phase, the preprocessing of the data, is done in Python.

3.2.1 *Discretisation of Signals*

The first step after reading the time series data from the Secure Water Treatment plant is to discretise the sample signals. An example of a signal received from the LIT101 sensor which measures the water tank level in subprocess 1 can be found in Figure 10. As in a state machine a process will be represented as a sequence of multiple events, it is essential to make a continuous signal discrete. This is necessary to determine when a system is in a similar state. In Figure 11 a part of the signal from an actuator (P101) is represented.

Before starting with the segmentation of the signals to create event sequences, some signals need to be denoised. As can be seen in Figure 10, there is still some noise present in the signal, which will cause the segmentation algorithm to create too many small segments. So to improve the segmentation results we will first denoise some of the signals by using a simple averaging filter. The signal after denoising can be found in Figure 12. In this case, the averaging filter is applied with a step of 70, which results in a decreased amount of samples

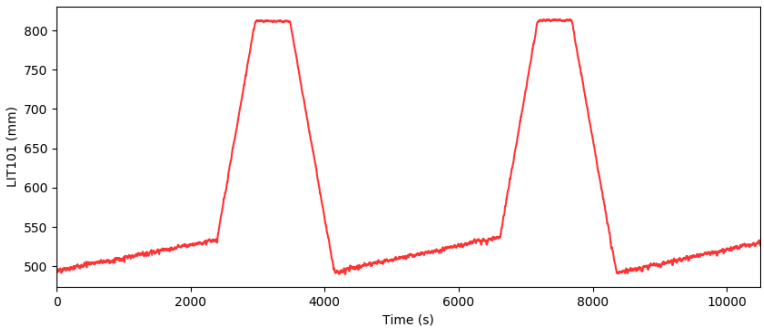

Figure 10. Signal from LIT101 sensor.
LIT101 is a level transmitter that measure the level of the raw water tank.

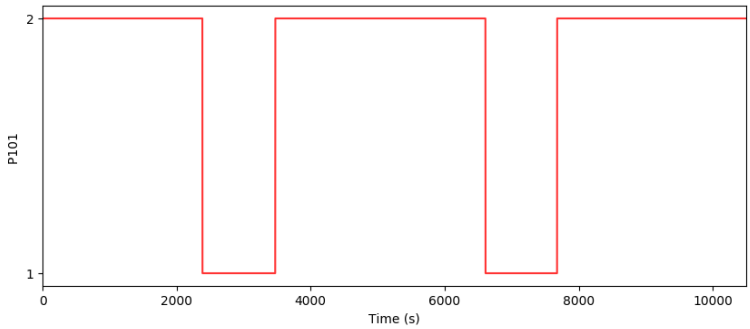

Figure 11. Signal from P101 actuator.
P101 is a pump that pumps water from the raw water tank to the second stage. In the plot '2' means *on* and '1' means *off*.

with factor 70. In the next stage the sample size will be set to normal in order to align multiple signals with the right time frame.

Online Segmentation Algorithm

By using a segmentation algorithm the signal will be split into different segments. To determine the segments we will make use of piecewise linear approximation (PLA). A sliding window approach is used similar as in [19]. A window will slide over the data points to apply linear interpolation, which represents the segments. The algorithm applies the interpolation for two data points, and proceeds by including the next data points until the calculated error of a potential segment exceeds the threshold which is manually specified beforehand and depends on the signal. This way the approximated segments can be determined without knowing the size of a segment beforehand. As this is an online algorithm it is not necessary to have a complete dataset beforehand, and therefore this can in the end be used for real-time attack detection. The used algorithm is based on the algorithm described in Keogh et al. [29].

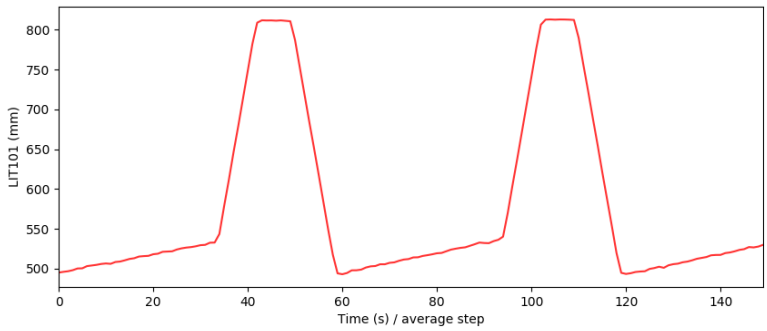

Figure 12. Denoised signal from LIT101 sensor.
The signal is denoised using an averaging approach. In this case with a step of 70.

Symbolic Representation

After the segmentation of the signals, a quantile-based discretisation function is used to group the segments into bins and give each bin a symbol. The grouping of the segments is based on the differential value of every segment. The amount of bins differ per signal and have to be specified manually by considering the type of signal. The symbols represent the events of a signal, with the final purpose of specifying the relation between multiple signals at a certain time frame.

In Figure 13 and Figure 14 the LIT101 and P101 signals are again visualised but this time they are represented with the segments and their symbols. For the signal of the LIT101 sensor this results in the following event sequence: 3 4 2 1 3 4 2 1 3. Such a string of symbols can be read into a state machine.

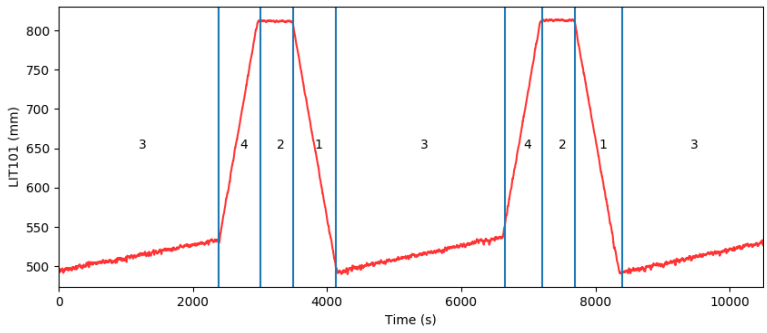

Figure 13. Segmented signal from LIT101 sensor.
A symbolic representation is given to the similar events. This results in an event sequence: 3 4 2 1 3 4 2 1 3. The grouping of the events is based on their differential value.

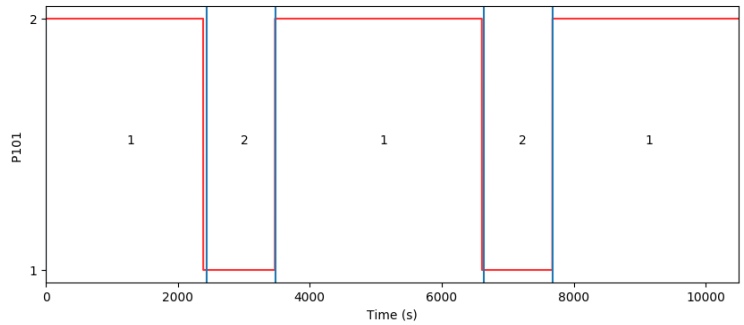

Figure 14. Segmented signal from P101 actuator.
A symbolic representation is given to the similar events. In the case of the pump there are only 2 different events, i.e. on and off.

3.2.2 *Timed Event Sequence*

A timed string is constructed by adding a time value for every element (symbol) in the sequence. The time value represents a time span, where a value $t_i \in \mathbb{N}$ for element e_i, is the time in seconds of the duration of an event (e_i, t_i).

The final result should be a timed string which we define as a sequence of events: $(e_1, t_1)(e_2, t_2)(e_3, t_3)...(e_n, t_n)$ where e_i is an input element and t_i a time value. The learning algorithm then learns an automaton from a set of these timed strings.

3.2.3 *Alignment of Signals*

In order to create a model of the behaviour of multiple signals, we first need to align the signals together. When taking a look at the definition of the general Mealy machine, it takes a pair (i/o) to move from one state to another. In this case it can also be a triple, quadruple etc., for example when aligning signals of four devices (e.g. D1/D2/D3/D4). We define this as a MIMO Mealy machine, where instead of two tapes and the relation between these two, there are in this case four different tapes.

The following example shows the aligned segments of the four devices in subprocess 1 for a part of the training data. See Table 9 in Appendix A for a list of all the devices. The alphabet of device D_0 contains four different symbols, the other devices D_1, D_2 and D_3 have an alphabet of two symbols. The different events in a signal were given numbers starting from 1.

The example also shows the time frame per event in seconds. As can be seen, the process has two full cycles in 8400 seconds, which is 2 hour and 20 minutes.

```
[0, 2380]       3/1/1/1
[2380, 3010]    4/1/2/1
[3010, 3500]    2/2/2/2
[3500, 4130]    1/2/1/2
[4130, 6650]    3/1/1/1
[6650, 7210]    4/1/2/1
[7210, 7700]    2/2/2/2
[7700, 8400]    1/2/1/2
[8400, 10850]   3/1/1/1
```

As mentioned before, the final input to the modeler should be a *timed event sequence*, so the last step here would be to transform the output after the alignment into a timed event sequence in the form of $(d1/d2/d3/d4, t)_1, (d1/d2/d3/d4, t)_2, ...(d1/d2/d3/d4, t)_n$. The time value t is in seconds and represents the duration of an event. This results in the following.

```
(3/1/1/1, 2380), (4/1/2/1, 630), (2/2/2/2, 490), (1/2/1/2, 630),
(3/1/1/1, 2520), (4/1/2/1, 560), (2/2/2/2, 490), (1/2/1/2, 700),
(3/1/1/1, 2450)
```

3.2.4 *Learning and Testing*

As can be seen in Figure 9, the sequencer phase has two outputs, of which one is acting as input to the modeler, and the other one is going directly to the detection phase. Both the training data as testing data are transformed into timed event sequences. The training data is used to learn the behaviour models of the SWaT testbed and the resulting sequences of the testing data are monitored for anomalies in the detection phase.

3.3 AUTOMATA LEARNING

Automata learning can be seen as *inferring* state machines or automata from data sequences. The models are created in two steps, first the RTI+ tool learns *real-time automata* (RTA) from the sequences, secondly these automata are transformed into Timed MIMO Mealy Machines (TMMM). This transformation is necessary because one automaton represents a subprocess that includes multiple signals that all need to be monitored for anomalous behaviour.

3.3.1 *Indirect Learning with RTI+*

The RTI+ tool that is used in this study stands for *real-time identification from positive data* and is written in C++ [30]. This tool is able to

identify a *probabilistic deterministic real-time automaton* (PDRTA) from positive data, i.e. monitored data of normal behaviour [22]. The algorithm is based on evidence-based state merging (EDSM). Statistical evidence is collected to decide to merge a pair of states or split a transition into two transitions. RTI+ is using a likelihood-ratio test as evidence.

Verwer et al. [22] define a real-time automaton as follows:

Definition 3.1 (RTA) *A real-time automaton is a 5-tuple* $A = \langle Q, \Sigma, \Delta, q_0, F \rangle$*, where* Q *is a finite set of states,* Σ *is a finite set of symbols,* Δ *is a finite set of transitions,* q_0 *is the start state, and* $F \subseteq Q$ *is a set of accepting states. A transition* $\delta \in \Delta$ *in an RTA is a tuple* $\langle q, q', a, [n, n'] \rangle$*, where* $q, q' \in Q$ *are the source and target states,* $a \in \Sigma$ *is a symbol, and [n, n'] is a delay guard.*

A PDRTA is a RTA which is *deterministic* meaning that there is only one transition given a symbol, source state and delay guard. Another characteristic that was added in the RTI+ tool is *probability*. Every transition will be assigned a probability, which is identified by counting the frequency of a sequence of events. The probability is used for classifying the test data. As previously mentioned, as only positive data is used for identifying the automata, there will be no accepting states in the PDRTA.

As in the end we want a Timed MIMO Mealy Machine of the normal behaviour of the SWaT testbed, this method can be seen as indirect learning by first identifying a PDRTA with RTI+, and next transform the PDRTA into a TMMM such that it can be used for anomaly detection.

3.3.2 *Algorithm RTI+*

The algorithm used for RTI+ can be found below. The state merging and transition splitting is using a *red-blue framework*. More information on RTI+ tool and the used framework can be found in Verwer et al. [22].

Algorithm 1 Real-time identification from positive data: RTI+

Require: A set of timed strings S+
Ensure: The result is a DRTA A
  Construct a timed prefix A tree from S+, color the start state q_0 of A red
  while A contains non-red states **do**
    Color blue all non-red target states of transitions with red source states
    Let $\delta =< q_r, q_b, a, g >$ be the most visited transition from a red to a blue state
    if the lowest p-value of a split is less than 0.05 **then**
      perform this split
    else if the highest merge p-value is greater than 0.05 **then**
      perform this merge
    else
      color q_b red
    end if
  end while

3.3.3 *Input to Modeler*

Similar as in Lin et al. [19] we create timed event sequences with a length of *two full process cycles*. This can be a different length of events in a sequence per subprocess as every subprocess can have a different amount of stages it goes through. All the testing data available is divided in these sequences which is the final input to the RTI+ tool. Every two successive sequences have an overlap of one cycle to make it easier for the tool to learn the loops in the behaviour [19].

3.3.4 *Output RTI+*

The following is an example output of RTI+.

```
0 1 [0, 2400]->1 #30 p=0.3125
0 1 [2401, 2460]->-1 #19 p=0.197917
0 1 [2461, 2640]->-1 #12 p=0.125
0 2 [0, 2640]->-1 #15 p=0.15625
0 3 [0, 2640]->-1 #15 p=0.15625
0 4 [0, 2640]->-1 #3 p=0.03125
0 5 [0, 2640]->-1 #1 p=0.0104167
0 6 [0, 2640]->-1 #1 p=0.0104167
1 2 [0, 2640]->2 #52 p=0.928571
1 3 [0, 2640]->-1 #4 p=0.0714286
2 3 [0, 2640]->3 #51 p=0.980769
2 4 [0, 2640]->-1 #1 p=0.0192308
3 1 [0, 2640]->-1 #1 p=0.0196078
3 4 [0, 2640]->4 #50 p=0.980392
4 1 [0, 2640]->1 #50 p=1
```

The first line can be read as starting state 0, reading symbol 1 with time guard $[0, 2400]$. These three values determine that the next state is 1 and the probability is 0.3125 where 30 sequences were observed that took this transition. The example shows that some transitions move to a state -1, which is called a *sink state*. This happens when

a sequence of events does not occur so often. Therefore a sinkstate arises as it was not able to merge with another existing state.

This textual version of the identified automaton is used for the anomaly detection, which is also done in C++. The tool also creates a graphical representation of the automaton for visualization. All the models can be found in Appendix B.

3.3.5 *Transformation*

As can be seen in the previous example, the symbol is just one value. As the RTI+ is not able to learn an automaton with more than one tape, i.e. a Mealy machine, it reads for example '3/1/1/1' as one symbol. That is why we need to 'transform' or basically decompose the symbol such that all the values can be used for the next phase.

3.4 ANOMALY DETECTION USING AUTOMATA

The final phase is the *anomaly detection* phase where the learned TM-MMs are used as a *one-class classifier* to classify the testing data as normal behaviour or anomalous behaviour.

3.4.1 *Classification*

In the detection phase the monitored sequences will be classified as normal behaviour or as anomalous behaviour according to the learned models. After creating an *error scoring list* for a monitored sequence, threshold values are used to determine if an alarm will be triggered, thus classified as anomalous, or not.

The error scoring list can be seen as *prioritising* the detected anomalies. As there is still some noise in the signals it is not feasible to trigger an alarm for every small anomaly that is observed. Therefore every event in a sequence will be given an error score between 0 and 1. While running a sequence through the learned state machine, an event is getting score 0 if the next transition can be fired perfectly given *the current state of the system, the symbols that are read, and the right timing*. If the next transition cannot be fired due to invalid symbols, wrong timing or the next state is a sink state, then this event will be given a score between 0 and 1 depending on the type and quantity of anomalies.

In addition, we consider the probability of a transition. It is preferred to see behaviour that is most likely to happen according to the model. When looking again at the output of RTI+ in Section 3.3.4, it shows that often when the probability is lower than 0.2, the transition moves to a sink state (-1). We use the probability in the transitions

to check which behaviour is most likely to occur and only use the transitions that have at least a probability of 0.2.

The error score is calculated by comparing what the sequence should have looked like according to the model. This is different than in TABOR [19], where as soon as a read symbol in the sequence could not be fired for transition, the whole sequence was classified as an anomaly. A model in TABOR represents the behaviour of a single sensor, where our models represent the behaviour of a subprocess, which can lead to more anomalies due the multiple devices that are involved. In the end every sequence will have an error list with a score between 0 and 1 for every event in this sequence.

3.4.2 *Evaluating Performance*

For evaluating the performance of the classification we need the ground truth labels of the testing sequences. In the time series dataset every *sample* is labeled as *Attack* or *Normal*. Whether the label of a *sequence* is attack or normal, is determined by the events in this sequence. If there is at least one attack sample present in an event, the whole event is labeled as attack. This results in a list with labels for every sequence, e.g. [000011100] where 0 means a normal event and 1 is an attack event. Finally, a sequence gets the label attack if there is at least one attack event (1) in the sequence. This is because the shortest attack takes only 10 minutes, which means that the attack frame can fall within a time frame of a single event. In the end, these ground truth labels are used for the performance measurement and are compared to the predicted labels which are the result of the classification, i.e. if a sequence got rejected by the TMMM, check if the the sequence was also labeled as '*Attack*'.

See Table 2 for the confusion matrix that is used in this study. In an early testing phase the results showed a lot of false positives, meaning a lot of falsely detected anomalies, which makes an IDS less efficient for final use. An important objective when creating an attack detection approach is to find the right balance between the false positives and the true positives. We want to minimize the amount of false alarms but simultaneously also be able to detect actual anomalous behaviour. For this purpose the thresholds are used as they are set to determine whether a sequence will trigger an alarm.

The thresholds are set per model by having multiple iterations and selecting the threshold that gives the best performance. More on the thresholds can be found in Section 4.2.2. The following performance metrics are used:

PRECISION Amount of correctly detected anomalous behaviour among all detected anomalous behaviour: $\frac{TP}{TP+FP}$

RECALL Amount of correctly detected anomalous behaviour among all anomalous behaviour: $\frac{TP}{TP+FN}$

F MEASURE $2 \cdot \frac{precision \cdot recall}{precision+recall}$

ACCURACY Percentage of correctly classified behaviour sequences: $\frac{TP+TN}{TP+TN+FP+FN}$

Actual behaviour	Detected behaviour	
	Normal	Anomaly
Normal	True Negative (TN)	False Positive (FP)
Anomaly	False Negative (FN)	True Positive (TP)

Table 2. Confusion matrix.
Metrics used for the performance of a classification model. In this study we define the positive data as the anomalous behaviour.

4

LEARNING BEHAVIOUR OF THE SWAT TESTBED

This chapter shows the results of the model learning. As explained in Chapter 3, the time series data that was collected from the testbed is transformed into timed event sequences and subsequently, the signals are put together per process. This chapter elaborates more on the different type of signals in a model and the difficulties that arose from those differences. It shows the input of the modelling tool which are the aligned signals, and the output: the learned models. Lastly, the performance of the model learning is discussed.

4.1 MODEL LEARNING

For the model-based anomaly detection approach six models of the six subprocesses are learned. We include all the devices per subprocess (see Table 3). All the devices with a description can be found in Table 9 in Appendix A.

Model	Included in automaton		Only value monitoring
	Sensors	Actuators	
1	LIT101, FIT101	P101, MV101	
2	FIT201	MV201, P201, P203, P205	AIT201, AIT202, AIT203
3	LIT301, FIT301, DPIT301	P302, MV301-MV304	
4	LIT401, FIT401	P402, P403, UV401	AIT401, AIT402
5	FIT501-FIT504	P501	AIT501-AIT504, PIT501-PIT503
6	FIT601	P601, P602, P603	

Table 3. Devices per model.
We define six models, one for every subprocess. As for some devices it is not possible to identify a symbolic sequence of the signals, these are only checked if they exceed the minimum and maximum threshold.

The water level sensors (LIT) and differential pressure sensors (DPIT) show clear sequential behaviour, therefore they can easily be used for creating the behaviour models (Fig. 15). Unlike the signals of the AIT and PIT sensors, which are not showing any repetitive behaviour.

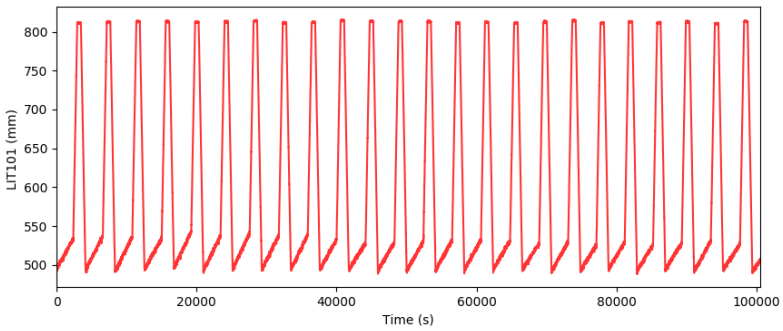

Figure 15. Signal from LIT101 sensor.
The signal from LIT101 shows clear stationary behaviour.

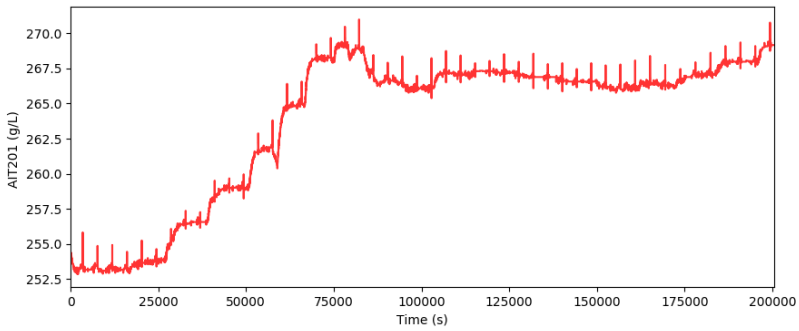

Figure 16. Signal from AIT201 sensor.
This sensor shows non-stationary behaviour, which leads to difficulties when segmenting the signal.

This makes it difficult to create a state machine of such a signal as in the pre-processing phase of the data it is already hard to define the signal as a sequence of events. An example of such a signal can be found in Figure 16. Therefore, in addition to learning the sequence behaviour, also the values of signals will be monitored. A minimum and maximum threshold value for the signals is determined and used for monitoring. Naturally, this is done for all the signals. Table 3 shows from which devices it was not possible to learn the sequence behaviour and thus were only used for value monitoring.

4.1.1 *Input of the Modeler*

For the first learned model the behaviour of the devices of subprocess 1 are included, which are: LIT101/FIT101/P101/MV101. All the signals are transformed into timed event sequences. The four signals including their segments can be found in Figure 17. All the signals of subprocess 1 will be aligned to create the final event sequence which is the input of the learning algorithm. As can be seen in Figure 18, the segments of the last three signals are aligned with the segments

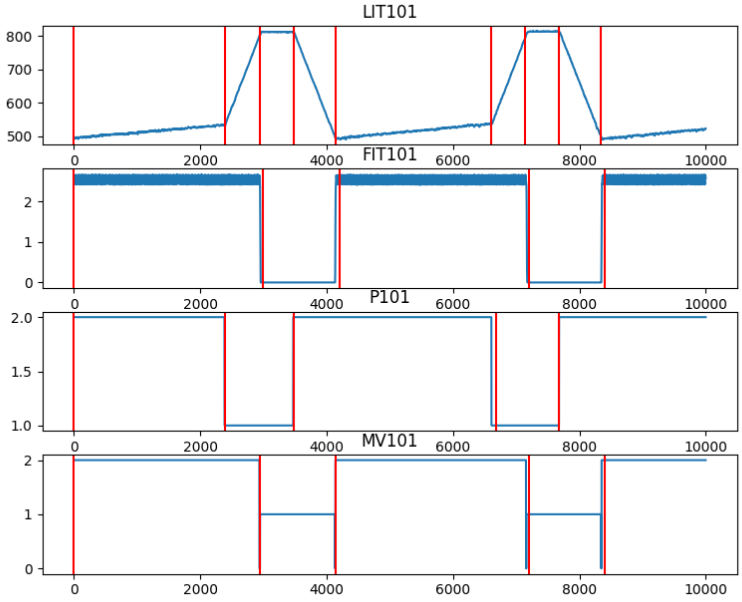

Figure 17. Segmented signals from Model 1.
The signals of the four devices in subprocess 1, LIT101, FIT101, P101 and MV101, are segmented. You can see that the LIT101 signal defines four different events. The other three devices only differentiate between two events. For example pump P101 can be on or off.

of the first signal. Then the final step for creating the sequences is to combine the symbols of the events together with the right time frames.

Similarly this is done for subprocesses 2-4. However, subprocess 5 and 6 gave some issues. As can be seen in Table 3, subprocess 5 includes the behaviour of many devices. Due to FIT sensors in this subprocess showing non-stationary behaviour and a lot of noise, the segmentation of the signals was already a problem. Symbols could not be correctly assigned to the segments and consequently, the signals could not be correctly aligned. A similar problem occurs in subprocess 6. The system is not able to correctly create the timed event sequences for this subprocess as all the signals of the pumps stay constant and in addition, the flow meter (FIT601) shows non-stationary behaviour.

For both subprocess 5 and 6 it was not possible to differentiate between the events and thus not able to assign the symbols to the segments. As a result, all events got the same symbol.

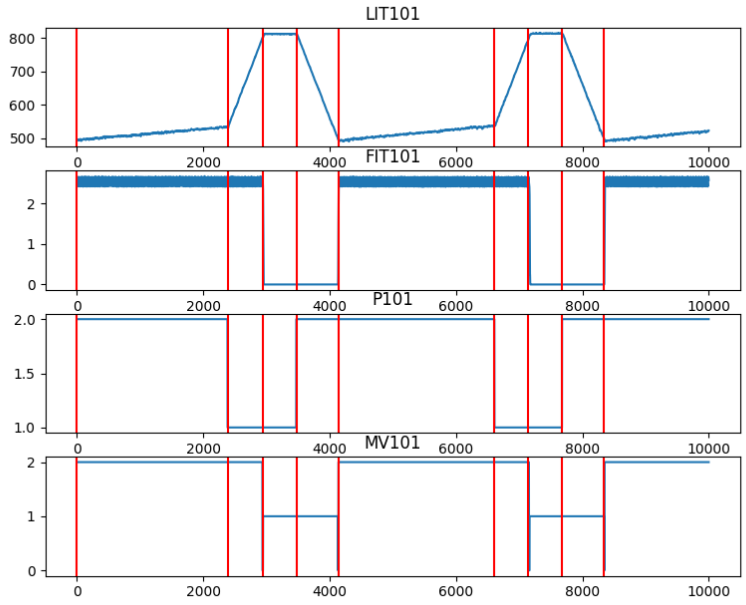

Figure 18. Aligned signals from Model 1.
The events of the signals are aligned according to the first signal, i.e. the LIT101 signal. This is because this signal changes most often to a different event.

4.1.2 *Output of the Modeler*

The resulting state machine for subprocess 1 is represented in Figure 19. This state machine is used for classifying the data sequences as normal behaviour or anomalous behaviour. All the learned models can be found in Appendix B. A transition in the model shows the symbols, timeframe, the probability and the amount of sequences that used this transition.

As for subprocess 5 and 6 we were not able to identify the sequence behaviour, all the events got the same symbol. The learned models of these two subprocesses are shown in Figures 31 and 32 in the Appendix. Although the learning tool was still able to create the state machines, they are probably not useful in detecting any sequence related anomalies. However, it is still possible to detect some anomalies in these processes as we also monitor the values of the signals to check if any thresholds are exceeded. The results of the anomaly detection can be found in Chapter 5.

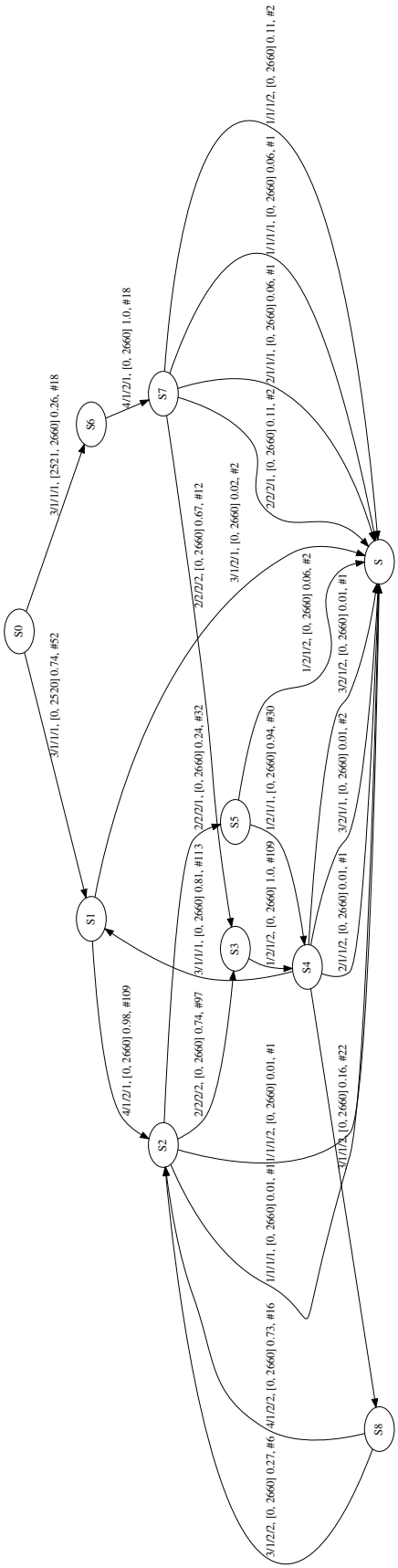

Figure 19. Trained model of subprocess 1.
Main loop according to the probabilities in the transitions: So - S1 - S2 - S3 - S4 - S1 etc. Transition shows symbols for all devices in subprocess, the time guard, the probability and the amount of observed training sequences taking this transition.

4.2 PERFORMANCE

This section discusses the performance of the model learning, i.e. the training phase. This includes the runtime, the amount of data that is necessary to create a behaviour model of a subprocess, and the evaluation of classifying the training data itself using the models.

4.2.1 *Computation*

Table 4 shows the runtime of the total training phase: learning six TMMMs. We compare this with three other studies that also used the SWaT testbed for testing a model-based anomaly detection approach [18], [19]. In addition, it took at average 28 seconds to read all the data of the 11 days (i.e. the training data and the testing data). As can be seen, the behaviour of the six subprocesses can be learned quite fast.

Method	DNN	SVM	TABOR	TMMM
Training runtime	2 weeks	30 min	214 s	41 s

Table 4. Runtime comparison of training phase.
Runtime comparison with studies from Inoue et al. [18] and Lin et al. [19].

4.2.2 *Evaluating Model with Training Data*

In a first testing phase, we see how the model performs on the training data itself. For this a procedure based on k-fold cross validation is used. As in time series data there are the temporal dependencies, it is not possible to create random sub training sets. In the original dataset the split in train set and test set is static, as only the last five days of the data contain the attacks.

For testing with the training data we can have multiple splits. In this case we use three splits. First iteration the model will be learned from the first 100.000 samples and is being tested with the next 100.000 samples. The next iteration the model is learned from the first 200.000 samples and tested with the next 100.000 samples etc. We calculate the *training error* for every split. The training error is defined as the percentage of sequences that are *incorrectly* classified.

The testing shows that when we do not use any threshold for the error lists, the training error is close to 1 (i.e. the accuracy is close to 0), meaning that almost all 'normal behaviour' training sequences are incorrectly classified as anomalous.

The cause here is that there is a lot of noise available in the signals. We observe that the RTI+ tool filters out the noise (the unlikely behaviour) by making use of sink states and probabilities, which makes

the models quite inflexible, i.e. not much leeway. The timed event sequences that are run through the state machine for classification do still contain some noise which results in error lists shown in the example below. In an error list, values above 0 mean an anomalous event (see Sec. 3.4.1). One error list belongs to one sequence:

```
[0.5, 0.5, 0.0, 0.0, 0.25, 0.25, 0.0, 0.0, 0.0]
[0.25, 0.25, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.25, 0.25, 0.0]
[0.0, 0.0, 0.25, 0.25, 0.0, 0.0, 0.0, 0.0, 0.25]
[0.0, 0.0, 0.0, 0.0, 0.25, 0.25, 0.0, 0.0, 0.0]
```

When considering the example above, which shows 5 error lists as a result of classification, we observe that if all of these sequences will trigger an alarm, it results in many false positives. We can say that the learned models are *underfitted*. If every sequence containing some anomalous events, is interpreted as not 'accepted' by the model, there will be too many alarms. Therefore the thresholds are used to add some leeway to the models.

Thresholds

We specify the first threshold as $p \in [0, 1]$ that defines the *percentage* of anomalies in an error list. Figure 20 shows that when the threshold is 0, the accuracy is close to 0, which means that almost all sequences labeled as normal, will be classified as *anomalous*. When setting threshold p to 1, all normal sequences are correctly classified as *normal* by the model. Now it seems that the threshold p should be 1, as it results in the highest accuracy. However, it does not say anything about detecting anomalies as we do not have anomalous labeled behaviour in the training data.

That is why we decided to do the same for the test data, which contains both normal and anomalous labeled data. Figure 21 shows the results for testing with the test data with all the thresholds (i.e. 0.1, 0.2 etc.). Now it is visible that if p is 0.6, most sequences are classified correctly. Figure 21 shows the results for testing with model 1. Testing the other models resulted in the same threshold.

In addition we used a threshold h to measure the severity of the anomalies by counting the amount of 'high' scored anomalies. In the example of the error lists above that would be a score of 0.75. The threshold h was determined per model and depends on the length of the sequence (i.e. in this study $h \in [2..4]$).

With the thresholds we can basically prioritise how important an anomaly is. For now we want that these sequences with a *minimal* error list do not trigger an alarm and that normal behaviour data is indeed classified as normal. The thresholds were used in the classification phase as a minimum requirement to trigger an alarm.

Although we already showed the accuracy of the test data of model 1 in Figure 21, the evaluation of the performance of the anomaly detection can be found in Chapter 5.

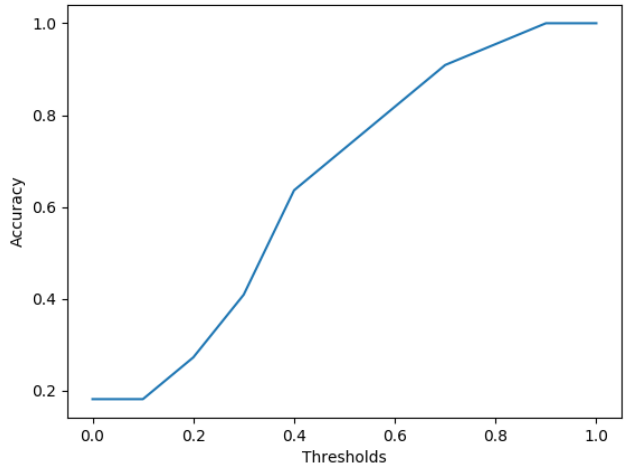

Figure 20. Performance of training data with different thresholds.
The figure shows that if there is no threshold (i.e. $p = 0$), the accuracy is close to 0, meaning that almost all normal sequences are classified as anomalous sequences. When increasing p the accuracy also increases. However, this only shows the performance for testing with a part of the training data itself.

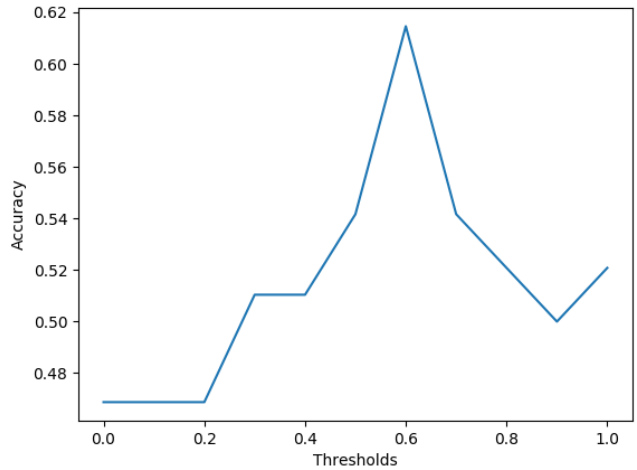

Figure 21. Performance testing data with different thresholds.
The figure shows a best performance when $p = 0.6$. These are the results when testing with model 1. The testdata includes both normal and attack behaviour.

4.2.3 *Minimum Input Size*

We trained the models with different amount of samples, for which a learning curve can be seen in Figure 22. The figure shows that at least 30.000 samples (~ 8 hours) were necessary to train the models of the SWaT testbed and get an acceptable performance. The performance is visualised as the accuracy when trying to classify a part of the training data itself. The accuracy is the percentage of correctly classified sequences.

We observed that, when training the model with 100.000, 200.000, 300.000 or 400.000 samples, there was a minimal to no difference in the state machines. Note that the minimum amount of samples obviously depends on the type of process, and that the minimum amount as shown in Figure 22 is specific to the SWaT testbed.

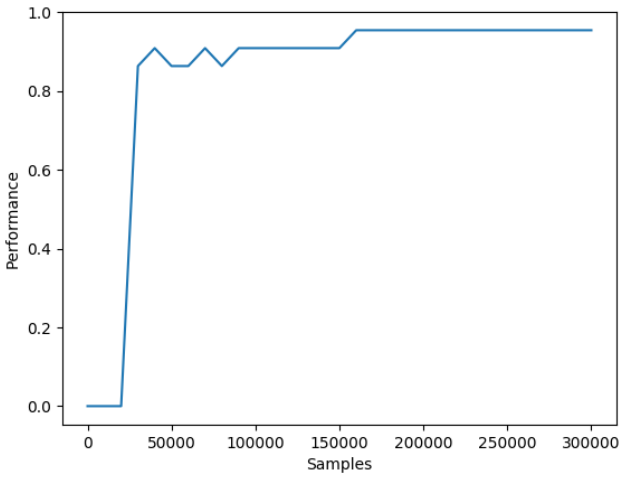

Figure 22. Learning curve for different amount of input samples.
The performance is defined as the percentage of correctly classified sequences. The curve shows that there is not a big difference in performance as long as there are more than 30.000 data samples used to learn the behaviour model. However, in this example we only tested with the training data. This means that although the model is able to classify normal behaviour correctly, we still do not know how it performs on anomalous behaviour data.

March, 2020

5 ANOMALY DETECTION USING BEHAVIOURAL MODELS

This chapter shows the results of the anomaly detection using the six behaviour models that were identified from the SWaT testbed. First we define the different types of anomalies, based on the study by Lin et al. [19], that are used in the attack detection approach. We discuss the performance of the anomaly detection and also the ability of this approach to detect the 36 attacks that were launched on the testbed.

5.1 TYPES OF ANOMALIES

The final *anomaly-based attack detection* differentiates between four types of anomalies:

SEQUENCE ANOMALY According to the current state, invalid symbols are read, meaning the state machine cannot move to the next state;

STATE ANOMALY State machine reaches a sink state, meaning that it is not possible to continue from this state on;

TIMING ANOMALY Given the current state, the duration of the event is not within the corresponding time guard;

VALUE ANOMALY When a signal is exceeding a certain threshold.

The first three types are all detected by the learned state machine. In addition, the signals are monitored and checked if they do not exceed any thresholds (i.e. value anomalies). Using these different types of anomalies the final attack detection system can indicate what is happening in the system during the time of an attack, e.g. which devices are behaving anomalously, does the system stays in a state too long or too short, or does the system ends in an unknown state.

5.2 DETECTING ANOMALOUS BEHAVIOUR

The performance of the classification of the test data per model can be found in Table 5. This table shows the results of detecting anomalies only using the state machines, thus the monitoring of the values of the signals - the value anomalies - is not included here. The test data is transformed into sequences, and every sequence will be classified, as it is not possible to check if a single data point or a single event is part of the state machine.

Model	TP	FP	FN	TN	Precision	Recall	Accuracy	F
1	13	6	35	42	0.684	0.271	0.572	0.388
2	2	4	45	55	0.333	0.043	0.538	0.075
3	6	2	32	27	0.75	0.158	0.493	0.261
4	7	0	41	81	1.0	0.146	0.717	0.254
5	*0*	*0*	*9*	*0*	-	-	-	-
6	0	0	56	75	0.0	0.0	0.573	0.0

Table 5. Performance of the separate models.
This table shows the performance of classifying the test sequences using the models. This means that test sequences created for subprocess 1, are classified using model 1 etc. The results of subprocess 5 are not a good representation of the performance, as there were already difficulties in the sequencer phase. The high FN rates are due to every model only representing a part of the behaviour of a system, and thus not every model is able to detect all anomalous behaviour.

The metrics that are used in Table 5 can be found in Section 3.4.2. The metrics from the confusion matrix (Table 2) can be defined as:

TP Correctly classified anomalous sequence

FP Normal sequence classified as anomalous

FN Anomalous sequence classified as normal

TN Correctly classified normal sequence

When looking at the performance metrics per model, the recall rates are quite low, i.e. low TP and high FN. The high false negative values make it seem that a lot of anomalous behaviour is not detected.

The cause of the low recall values is the fact that *every model only represents a sub part of the behaviour of the system* and thus every model is able to detect different anomalous behaviour. Therefore, even if a model is not able to detect an anomalous sequence, another model might. See Tables 10 and 11 for the attack scenarios which cause the anomalous behaviour. The tables show which devices and subprocesses they affect.

We observe that when considering the definition of recall, in this study, *all anomalous behaviour* is difficult to define per model. We can say that the recall is this low because it does not take into account that not all attacks can be identified in every model, as an attack on the testbed not necessarily affects all the subprocesses.

Consequently, as the true positives (the correctly detected anomalies) will not be high per separate model, the main objective here was to minimize the amount of false positives to make the attack detection

as effective as possible. As already mentioned before, false alarms in industrial control systems come at great cost.

Using thresholds mentioned in the previous chapters, the aim per model was to find the balance between TPs and FPs. As can be seen in Table 5 the ratio is different per model. Although the separate models do not have a good performance, the main goal is to identify the attacks with as little FPs as possible *when all the models are combined*, which can be found in Section 5.3.

The results of Model 5 are not really representative, which is partly due to the learned model. As can be seen in Table 3 in Chapter 4, Model 5 includes many devices, of which many do not have stationary behaviour. This makes it difficult to learn a model which is a good representation of the process behaviour. In addition, it appears that in the sequencer phase the test data was divided in only 9 sequences, which all have as ground truth label *Attack*. This is because the division of the testdata in 9 subsequences makes each sequence cover a large time frame, resulting in each sequence including at least one attack scenario. All the sequences were finally classified as normal behaviour, which in this context can be considered more practical then all of them causing an alarm. Although this model was unable to detect any sequence, timing and state anomalies, for the final detection it is still possible to detect some anomalies by monitoring the signals in subprocess 5.

Model 6 was not able to detect any anomalous behaviour which corresponds with the used dataset, as the authors mentioned they did not launch any attack on the devices in subprocess 6.

5.3 DETECTING THE ATTACK SCENARIOS

The final goal of the anomaly-based attack detection is to detect the anomalous behaviour that was caused by the 36 attacks that were launched during the collecting of the test data. A description of the 36 attack scenarios can be found in Table 10 in Appendix A. For the end result all models are combined and the signals will be monitored to check if they exceed any thresholds.

The evaluation of detecting the attack scenarios is done similarly as in Lin et al. [19] using a window-based method. For this evaluation the following metrics are defined:

- **TP** A correctly detected ground-truth attack scenario;
- **FP** A detection without an overlap with any ground-truth attack scenario;
- **FN** A ground-truth attack scenario that was not detected.

It is called a window-based evaluation since we consider if a detection frame overlaps the actual attack frame. Therefore, similar as

in Lin et al., we do not examine the true negatives as this evaluation is about detecting the attack scenarios, even if it detects too late, too early or with correct timing.

In total 28 of the 36 attack scenarios were detected, compared to 13, 20, and 24 scenarios for DNN, SVM [18] and TABOR [19], respectively. Table 11 in Appendix A shows per model which scenarios it can detect. The final evaluation can be found in Table 6 below.

Precision	Recall	F measure	TP	FP	FN
0.84848	0.77778	0.81159	28	5	8

Table 6. Performance of attack detection.
The final evaluation of the detected scenarios. This table shows the results when all the models are used for classification and also the signals are monitored using their values and thresholds. This leads to the detection of 28 scenarios and 5 false positives over 5 days.

After combining all the models, the attack detection triggers five false alarms in 5 days, which can be considered acceptable for this first proof of concept. With a precision of 0.84848 we could say that of all the alarms that this approach will trigger, around 85 percent is true positive.

Sometimes it appears that an attack is detected only after it already ended, which causes a false positive, which - if there was a ground truth scenario - can still help the human operator to find out what happened in the system. Another possibility, which can also be seen in the data, is that the system is still stabilising after an attack was launched on the system. The results showed that a majority of the models detect anomalous behaviour in similar time frames after attack 9. This can mean that processes are still recovering from the previous attacks.

A detection frame that already starts before the attack scenario itself does not mean that an attack is detected before it even started. As for real-time use, which is a final goal for this approach, the physical data will be the input and then checked for anomalous behaviour. Then a time frame will be given in which anomalous behaviour is observed, including which process and devices were involved. As in this study we check data from 5 days altogether, we see in the results where in the data the attack detection observes anomalous behaviour. However, when this should be used in real-time, and alarm will go off at the start of such a time frame.

5.3.1 *Examples of Detection Results*

Figure 23 shows a big detection frame (in red) for scenario 7 until 9. However, scenario was detected by model 3 and scenario 8 and 9 by model 4. This corresponds to the given scenarios, which can be

found in Table 10 in Appendix A. In scenario 7 the value of DPIT301 is changed, and in 8 and 9 the behaviour of FIT401 is altered.

The figure visualises the large frames because when a detected anomaly is of high priority, the whole sequence will be classified as anomalous. While there might be some normal events observed in between the attack events, they are not showing in the graph.

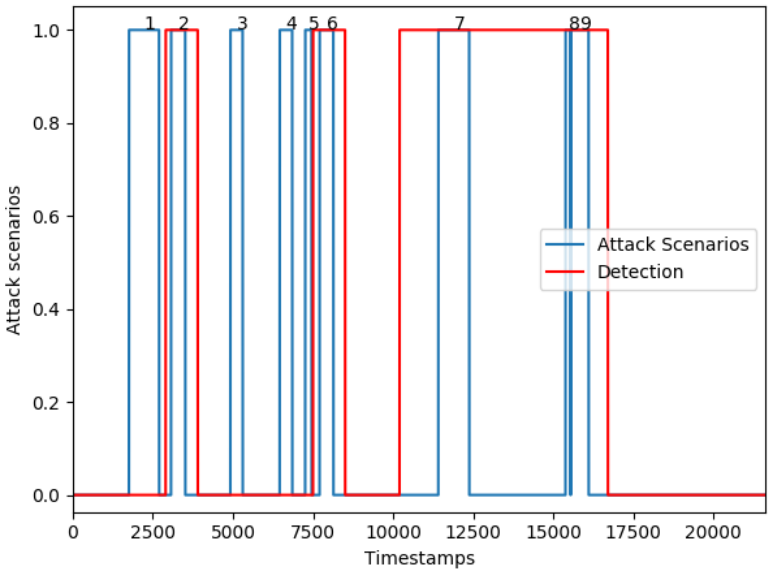

Figure 23. Detection of attack scenarios 1 until 9.
The graph shows a large detection frame for attack 7, 8 and 9. However, they are detected by different models. See Table 11 in Appendix A to see which model detects which scenario and on which process the attack was launched.

Figure 24 shows an example of a large detection frame which 'detects' scenario 10 until 16. They were all detected due to observed anomalous behaviour in subprocess 3. Although scenario 10-13 do indeed influence devices in subprocess 3, the other scenarios (14-16) are possibly detected 'by accident' because anomalous behaviour was observed due to recovery, or they indirectly also influence the third subprocess. However, scenarios 14 until 16 were also detected due to observed anomalous behaviour in subprocess 1. An issue here can be that although anomalous behaviour is observed, it does not necessarily give the right origin of the attack.

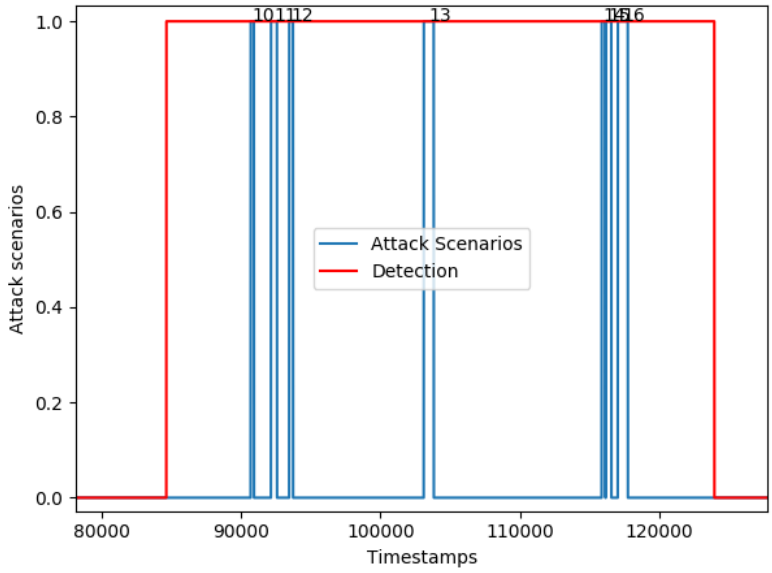

Figure 24. Detection of attack scenarios 10 until 16.
The graph shows a large detection frame, meaning that in this time frame continuous anomalous behaviour was observed.

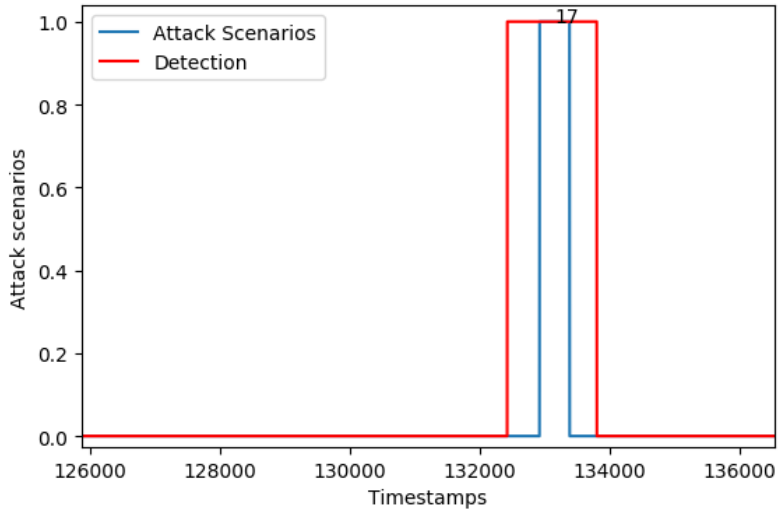

Figure 25. Detection of attack scenario 17.
Correct detection of attack scenario 17 that was launched on subprocess 4 and 5 (see Table 10) and also detected by model 4 and 5 (see Table 11).

Attack scenario 23 has a duration of almost 10 hours and closes off pump P302. Figure 26 illustrates the detection of this attack which is a quite accurate detection. It also detects attack scenario 22 which is also launched on the same device.

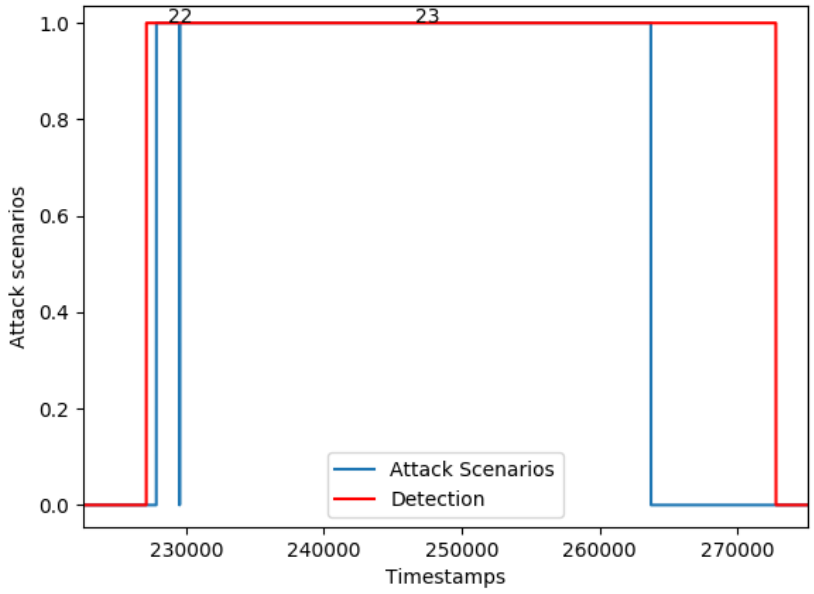

Figure 26. Detection of attack scenario 22 and 23.
Scenario 23 closed off pump P302 and took almost 10 hours. This attack was detected by almost all models (see Table 11).

5.4 COMPARISON WITH RELATED LITERATURE

When comparing the performance with TABOR [19], SVM and DNN [18], our TMMM approach shows around the same performance. Similar as TABOR this approach shows a few more false positives but also a higher recall, meaning that more attack scenarios are detected. The performance results can be found in Table 7.

Method	Precision	Recall	F measure
TABOR	0.86171	0.78803	0.82322
DNN	0.98295	0.67847	0.80281
SVM	0.92500	0.69901	0.79628
TMMM	0.84848	0.77778	0.81159

Table 7. Evaluation comparison.
Comparison of final evaluation of the detection of the attack scenarios with studies from Inoue et al. [18] and Lin et al. [19].

The runtime is a little slow compared to TABOR [19] (Table 8). This is mainly due to the creation of the testing sequences, the comparison between the detection results and the ground-truth scenarios for measuring the performance, and the checking of the values of the signals. However, we observed that the classification phase, where

all the testing sequences are run through the model, only takes *0.2 seconds*. Which is quite fast for 5 days of data.

For the final attack detection approach the runtime should be faster, as the testing will not be done for 5 days altogether.

Method	DNN	SVM	TABOR	TMMM
Testing runtime	8 hours	10 min	33 s	86 s

Table 8. Runtime comparison of testing phase.
Runtime comparison with studies from Inoue et al. [18] and Lin et al. [19].

6 CONCLUSION

This study shows the efficiency of an attack detection approach using models of the normal behaviour of the physical process of an industrial control system. The models were automatically identified without needing expert knowledge on the system in less than a minute time. Time series data of the sensors and actuators were combined together per subprocess and used to identify a TMMM model. In the end we had six TMMMs that represent the behaviour of the SWaT testbed. The learned TMMMs are useful to get a better understanding of the processes, as they give insight into the different stages of a subprocess and their duration. By making use of the visualisation and the prioritisation of anomalies, this approach also improves the efficiency for an operator of a monitoring system. In the end we were able to create four good models of the first four subprocesses. However, the last two models were not giving a good representation of the subprocesses and were also not usable for anomaly detection. Overall the models were still able to detect 28 out of 36 attack scenarios and the performance showed a final precision rate of 0.85.

Compared to the study from Lin et al. [19], where they only create timed automata of some of the sensors and in addition need to learn the dependencies between devices, our created TMMMs give insight into how the signals in a subprocess are behaving together without the need of combining multiple types of models. For future purposes, when being used in real-time, this approach is able to show in which subprocess, which device including signal values and in which time frame anomalous behaviour is present.

When considering our first sub research question, *the feasibility to map an ICS process to a single model*, we observed that it depends a lot on the type of devices and the noise in the signals. A few devices, such as the pressure meters and sensors that analyse values such as the pH level or NaCl level, do not show any repetitive behaviour in the used dataset. It might be possible to learn trends in signals when more data is available, i.e. observing the processes over a longer time. However, as many attacks include spoofing the values of such sensors it might be enough to just monitor these values instead of learning the sequence behaviour.

Nevertheless, the models of the first four subprocesses were, without any complications, automatically identified from the time series data. As can also be seen in Section 2.1.1, these four subprocesses have a clear and repeating task involving a few actuators and a few sensors. As compared to subprocess 5, which includes mostly sensors

operating in the RO unit, i.e. a purification process. This can explain the difficulty to identify the TMMM of subprocess 5. Furthermore, subprocess 6 involves the cleaning which seems to be not active during the data collection, as the values of the pumps involved, stayed at a constant level. Hence, model 6 was not usable in the end.

We observed that the difficulties with learning models 5 and 6 started with the pre-processing of the corresponding signals of subprocesses 5 and 6. The sliding window algorithm for segmentation was able to detect segments in the non-stationary time series data. However for the next step, where similar segments are grouped together based on their differential value, it was not possible to detect similar trends in the signals. Even for the stationary signals it sometimes occurred that symbols were assigned incorrectly. Although it was not part of the scope of this study to improve the pre-processing of the signals, it is an important research topic in related literature. For future improvements there could be more focus on the signal processing phase.

The results in Chapter 4 also showed that when too many devices are involved in a subprocess - of which many also show non-stationary behaviour - it becomes more difficult to learn an usable model. In future research, it might be possible to subdivide the groups of devices. This leads in the end to more models, but they would be more simple and conceivably also better in identifying anomalous behaviour. However, such a decision should be made manually and will also influence the visualisation of the complete behaviour.

A sub goal of this study was to learn the models without needing expert knowledge. We did have the process overview (Fig. 6) of the SWaT testbed of which we derived the six subprocesses, i.e. six models. For now we still needed to specify a few parameters for the segmentation and denoising of the signals. However, this can be automated in the future. In addition, we needed to set the thresholds for the classification. To determine the optimal thresholds we observed that it is necessary to have anomalous labeled data. For future and other cases it is important to use real or artificial attack data to test the models and set the optimal thresholds, which finally will result in better detection rates and less false alarms.

This leads to our second sub research question, *using the TMMMs to monitor the behaviour of an ICS*, where we looked at the capability to detect anomalous behaviour in the testbed and then particularly the anomalous behaviour that was caused by the 36 launched attacks. By making use of the aforementioned thresholds for the error lists, we are basically able to prioritise anomalies and reduce the amount of false positives. The separate TMMMs did not show a good performance (i.e. high FN rate), as it is most probable that a model only detects anomalous behaviour from the attacks launched on the corresponding subprocess. However, overall the proposed approach

showed a good performance. Of the 36 attacks scenarios, this approach detected 28 correctly and raised 5 false alarms in total spread over 5 days. When looking at the final precision of the attack detection we can say that of all the alarms the models trigger, around 85 percent is relevant.

As could be seen in Chapter 5, the detection results show detection frames in which more than one scenario was detected. Due to time constraints we did not yet visualise where the anomalous behaviour was exactly located when an alarm was triggered. However, in further research we can simulate the anomalous behaviour using the models to give more insights into the attacks. In this study we showed that models on an ICS could be learned efficiently without needing expert knowledge and that the models were able to detect anomalous behaviour caused by launched attacks. The proposed models can also be of great use for analysing attacks on industrial control systems by first modelling the complete behaviour, and then visualise the attack and see how it can affect a system.

March, 2020

A

TABLES

No.	Device	Type	Description
1	FIT101	Sensor	Flow meter; Measures inflow into raw water tank
2	LIT101	Sensor	Level Transmitter; Raw water tank level
3	MV101	Actuator	Motorized valve; Controls water flow to the raw water tank
4	P101	Actuator	Pump; Pumps water from raw water tank to second stage
5	P102	(backup) Actuator	Pump; Pumps water from raw water tank to second stage
6	AIT201	Sensor	Conductivity analyser; Measures NaCl level
7	AIT202	Sensor	pH analyser; Measures HCl level
8	AIT203	Sensor	ORP analyser; Measures NaOCl level
9	FIT201	Sensor	Flow Transmitter; Control dosing pumps
10	MV201	Actuator	Motorized valve; Controls water flow to the UF feed water tank
11	P201	Actuator	Dosing pump; NaCl dosing pump
12	P202	(backup) Actuator	Dosing pump; NaCl dosing pump
13	P203	Actuator	Dosing pump; HCl dosing pump
14	P204	(backup) Actuator	Dosing pump; HCl dosing pump
15	P205	Actuator	Dosing pump; NaOCl dosing pump
16	P206	(backup) Actuator	Dosing pump; NaOCl dosing pump
17	DPIT301	Sensor	Differential pressure indicating transmitter; Controls the backwash process
18	FIT301	Sensor	Flow meter; Measures the flow of water in the UF stage
19	LIT301	Sensor	Level Transmitter; UF feed water tank level
20	MV301	Actuator	Motorized Valve; Controls UF-Backwash process
21	MV302	Actuator	Motorized Valve; Controls water from UF process to De-Chlorination unit
22	MV303	Actuator	Motorized Valve; Controls UF-Backwash drain
23	MV304	Actuator	Motorized Valve; Controls UF drain
24	P301	(backup) Actuator	UF feed Pump; Pumps water from UF feed water tank to RO feed water tank via UF filtration
25	P302	Actuator	UF feed Pump; Pumps water from UF feed water tank to RO feed water tank via UF filtration
26	AIT401	Sensor	RO hardness meter of water
27	AIT402	Sensor	ORP meter; Controls the $NaHSO_3$dosing(P203), NaOCl dosing (P205)
28	FIT401	Sensor	Flow Transmitter; Controls the UV dechlorinator
29	LIT401	Actuator	Level Transmitter; RO feed water tank level
30	P401	(backup) Actuator	Pump; Pumps water from RO feed tank to UV dechlorinator
31	P402	Actuator	Pump; Pumps water from RO feed tank to UV dechlorinator
32	P403	Actuator	Sodium bi-sulphate pump
33	P404	(backup) Actuator	Sodium bi-sulphate pump
34	UV401	Actuator	Dechlorinator; Removes chlorine from water
35	AIT501	Sensor	RO pH analyser; Measures HCl level
36	AIT502	Sensor	RO feed ORP analyser; Measures NaOCl level
37	AIT503	Sensor	RO feed conductivity analyser; Measures NaCl level
38	AIT504	Sensor	RO permeate conductivity analyser; Measures NaCl level
39	FIT501	Sensor	Flow meter; RO membrane inlet flow meter
40	FIT502	Sensor	Flow meter; RO Permeate flow meter
41	FIT503	Sensor	Flow meter; RO Reject flow meter
42	FIT504	Sensor	Flow meter; RO re-circulation flow meter
43	P501	Actuator	Pump; Pumps dechlorinated water to RO
44	P502	(backup) Actuator	Pump; Pumps dechlorinated water to RO
45	PIT501	Sensor	Pressure meter; RO feed pressure
46	PIT502	Sensor	Pressure meter; RO permeate pressure
47	PIT503	Sensor	Pressure meter; RO reject pressure
48	FIT601	Sensor	Flow meter; UF Backwash flow meter
49	P601	Actuator	Pump; Pumps water from RO permeate tank to raw water tank
50	P602	Actuator	Pump; Pumps water from UF back wash tank to UF filter to clean the membrane
51	P603	Actuator	Not implemented in SWaT yet

Table 9. Actuators and sensors of the SWaT testbed.
The 51 devices of the SWaT testbed with description [24].

No.	Attack description
1	Open MV101
2	Turn on P102
3	Increase LIT101 by 1mm every second
4	Open MV504
5	Set the of AIT202 to 6
6	Water level LIT301 increased above HH
7	Set value of DPIT301 as >40kpa
8	Set value of FIT401 as <0.7
9	Set value of FIT401 as 0
10	Close MV304
11	Do not let MV303 open
12	Decrease water level LIT301 by 1mm each second
13	Do not let MV303 open
14	Set value of AIT504 to 16 uS/cm
15	Set value of AIT504 to 255 uS/cm
16	Keep MV101 on continuously; Value of LIT101 set as 700mm
17	Stop UV401; Value of AIT502 set as 150; Force P501 to remain on
18	Value of DPIT301 set to >0.4 bar; Keep MV302 open; Keep P602 closed
19	Turn off P203 and P205
20	Set value of LIT401 as 1000; P402 is kept on
21	P101 is turned on continuously; Set value of LIT301 as 801mm
22	Keep P302 on continuously; Value of LIT401 set as 600mm till 1:26:01
23	Close P302
24	Turn on P201; Turn on P203; Turn on P205
25	Turn P101 on continuously; Turn MV101 on continuously; Set value of LIT101 as 700mm; P102 started itself because LIT301 level became low
26	Set LIT401 to less than L
27	Set LIT301 to above HH
28	Set LIT101 to above H
29	Turn P101 off
30	Turn P101 off; Keep P-102 off
31	Set LIT101 to less than LL
32	Close P501; Set value of FIT502 to 1.29 at 11:18:36
33	Set value of AIT402 as 260; Set value of AIT502 to 260
34	Set value of FIT401 as 0.5; Set value of AIT502 as 140 mV
35	Set value of FIT401 as 0
36	Decrease LIT301 value by 0.5mm per second

Table 10. Description of attack scenarios.
The original dataset contains 41 attack scenarios, however only 36 have physical impact on the testbed. This table gives a description of these 36 scenarios [24].

Attack No.	Affected process	Detection by model				
		1	2	3	4	5
1	1					
2	1		x			
3	1					
4	5					
5	2					
6	3			x		
7	3			x	x	
8	4				x	x
9	4				x	x
10	3	x		x		
11	3	x		x		
12	3			x		
13	3			x		
14	5	x		x		
15	5	x		x		
16	1	x		x		
17	4,5				x	x
18	3,6			x	x	x
19	2				x	x
20	4				x	x
21	1,3	x	x			
22	3,4		x			
23	3		x	x	x	x
24	2					
25	1,3	x	x	x		
26	4					
27	3	x	x			
28	1	x				
29	1					
30	1					
31	1	x				
32	5	x				x
33	4,5	x				x
34	4,5	x			x	x
35	4	x			x	x
36	3			x		
True positives		14	6	13	10	11
False positives		4	3	3	0	0

Table 11. Final detection of attack scenarios.
Model 6 is not included in this table as this model did not detect any anomalous behaviour. The table shows that most of the attacks were detected by the model of the process of which the attack was launched on. Some attacks were also visible in the following processes. Although model 5 did not detect any sequence anomalies, it did observe many anomalous behaviour in the values of the signals.

No.	Data No.	Start time	End time	Samples	Type
1	1	28/12/2015 10:29:14	10:44:53	[1755, 2694]	SSSP
2	2	28/12/2015 10:51:08	10:58:30	[3069, 3511]	SSSP
3	3	28/12/2015 11:22:00	11:28:22	[4921, 5303]	SSSP
4	4	28/12/2015 11:47:39	11:54:08	[6460, 6849]	SSSP
	5	28/12/2015 11:58:20	No Physical Impact Attack		
5	6	28/12/2015 12:00:55	12:04:10	[7256, 7451]	SSSP
6	7	28/12/2015 12:08:25	12:15:33	[7706, 8134]	SSSP
7	8	28/12/2015 13:10:10	13:26:13	[11411, 12374]	SSSP
-	9	28/12/2015 14:15:00	No Physical Impact Attack		
8	10	28/12/2015 14:16:20	14:19:00	[15381, 15541]	SSSP
9	11	28/12/2015 14:19:01	14:28:20	[15542, 16101]	SSSP
-	12	29/12/2015 11:10:40	No Physical Impact Attack		
10	13	29/12/2015 11:11:25	11:15:17	[90686, 90918]	SSSP
11	14	29/12/2015 11:35:40	11:42:50	[92141, 92571]	SSSP
-	15	29/12/2015 11:52:01	No Physical Impact Attack		
12	16	29/12/2015 11:57:25	12:02:00	[93446, 93721]	SSSP
13	17	29/12/2015 14:38:12	14:50:08	[103093, 103809]	SSSP
-	18	29/12/2015 18:08:55	No Physical Impact Attack		
14	19	29/12/2015 18:10:43	18:15:01	[115844, 116102]	SSSP
15	20	29/12/2015 18:15:43	18:22:17	[116144, 116538]	SSSP
16	21	29/12/2015 18:30:00	18:42:00	[117001, 117721]	SSMP
17	22	29/12/2015 22:55:18	23:03:00	[132919, 133381]	MSMP
18	23	30/12/2015 01:42:34	01:54:10	[142955, 143651]	MSMP
19	24	30/12/2015 09:51:08	09:56:28	[172269, 172589]	SSMP
20	25	30/12/2015 10:01:50	10:12:01	[172911, 173522]	SSMP
21	26	30/12/2015 17:04:56	17:29:00	[198297, 199741]	MSSP
22	27	31/12/2015 01:17:08	01:45:18	[227829, 229521]	MSSP
23	28	31/12/2015 01:45:19	11:15:27	[229522, 263728]	SSSP
24	29	31/12/2015 15:32:00	15:34:00	[279121, 279241]	SSMP
25	30	31/12/2015 15:47:40	16:07:10	[280061, 281231]	MSMP
26	31	31/12/2015 22:05:34	22:11:40	[302654, 303020]	SSSP
27	32	1/01/2016 10:36:00	10:46:00	[347680, 348280]	SSSP
28	33	1/01/2016 14:21:12	14:28:35	[361192, 361635]	SSSP
29	34	1/01/2016 17:12:40	17:14:20	[371480, 371580]	SSSP
30	35	1/01/2016 17:18:56	17:26:56	[371856, 372336]	SSMP
31	36	1/01/2016 22:16:01	22:25:00	[389681, 390220]	SSSP
32	37	2/01/2015 11:17:02	11:24:50	[436542, 437010]	SSMP
33	38	2/01/2015 11:31:38	11:36:18	[437418, 437698]	MSSP
34	39	2/01/2015 11:43:48	11:50:28	[438148, 438548]	MSSP
35	40	2/01/2015 11:51:42	11:56:38	[438622, 438918]	SSSP
36	41	2/01/2015 13:13:02	13:40:56	[443502, 445191]	SSSP

Table 12. Attack Scenarios details.
The attack scenarios from the dataset including their duration and type. The second column represents the original numbering of the scenarios. Five scenarios were excluded from this research as they did not have any physical impact on the testbed [24].

March, 2020

B

MODELS

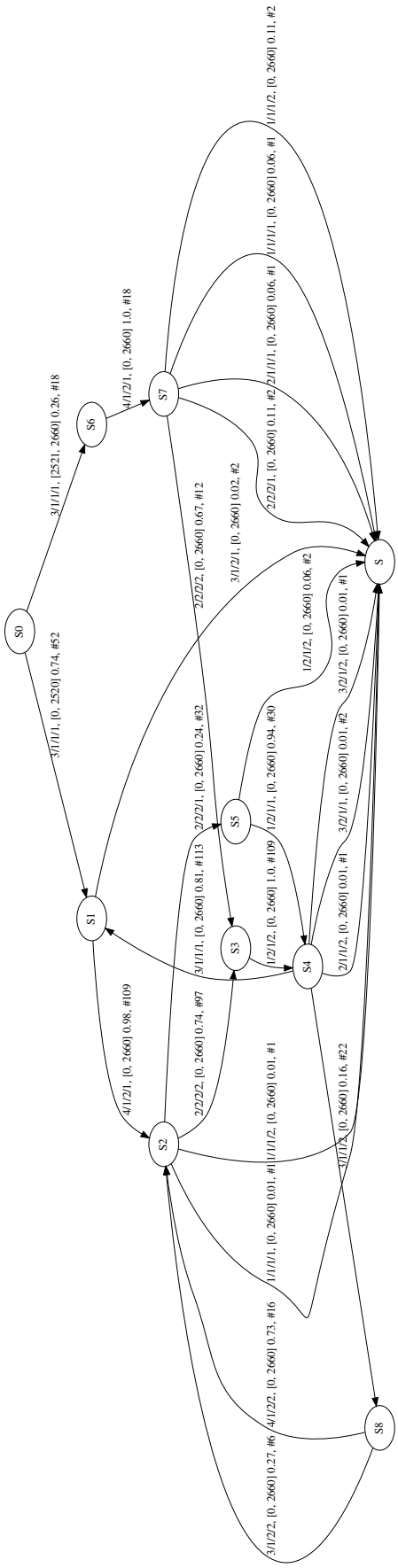

Figure 27. Trained model of subprocess 1.

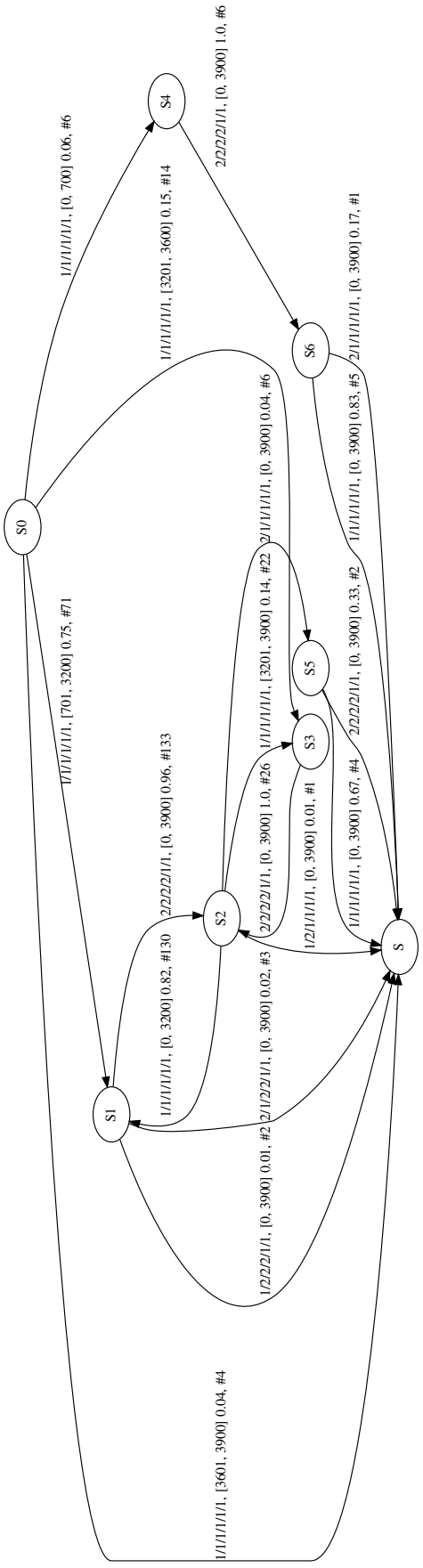

Figure 28. Trained model of subprocess 2.

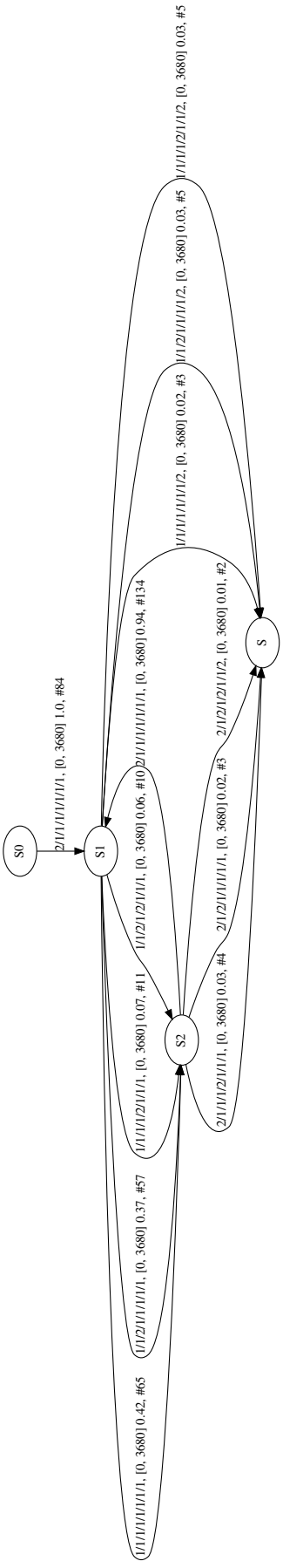

Figure 29. Trained model of subprocess 3.

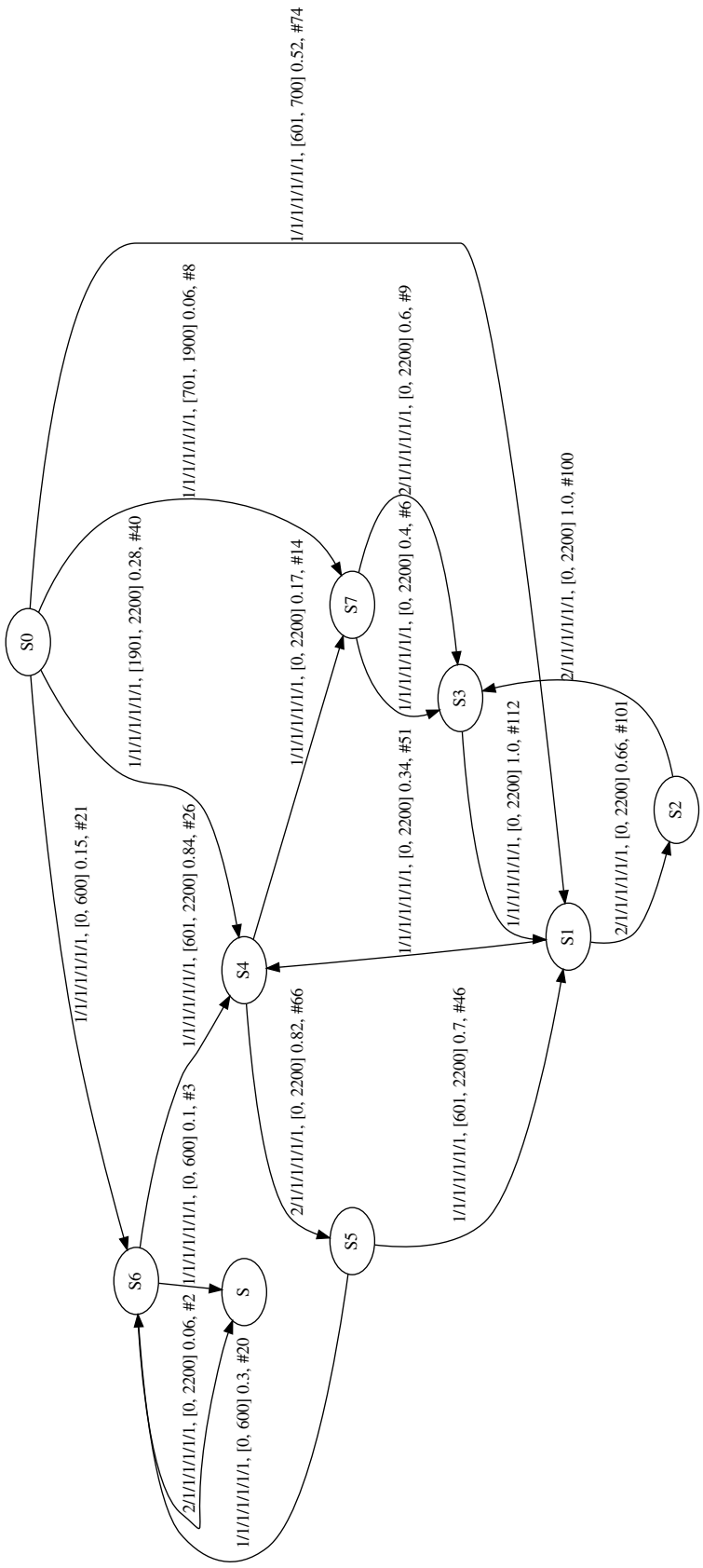

Figure 30. Trained model of subprocess 4.

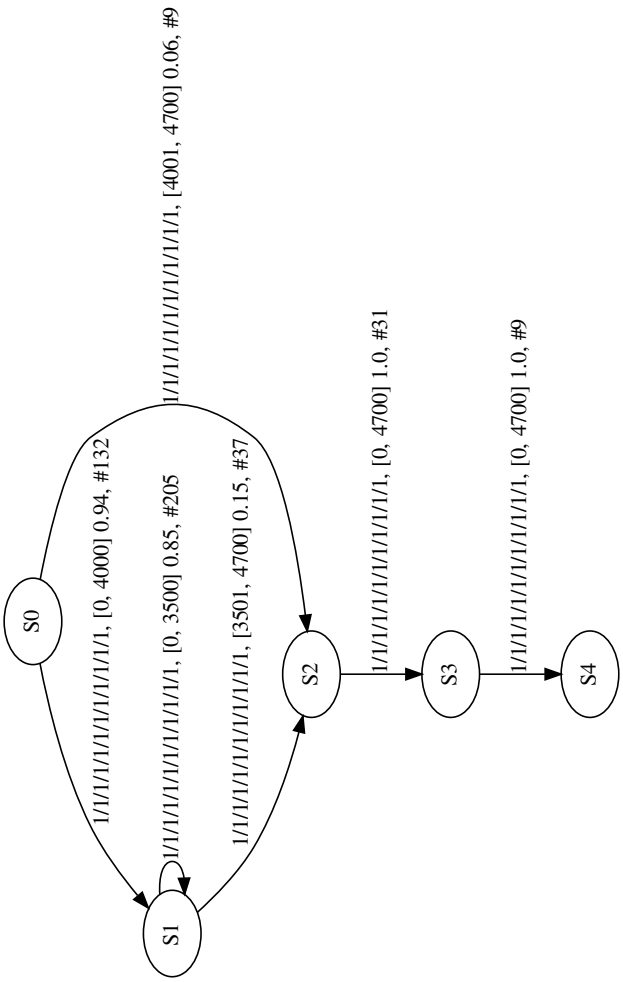

Figure 31. Trained model of subprocess 5.

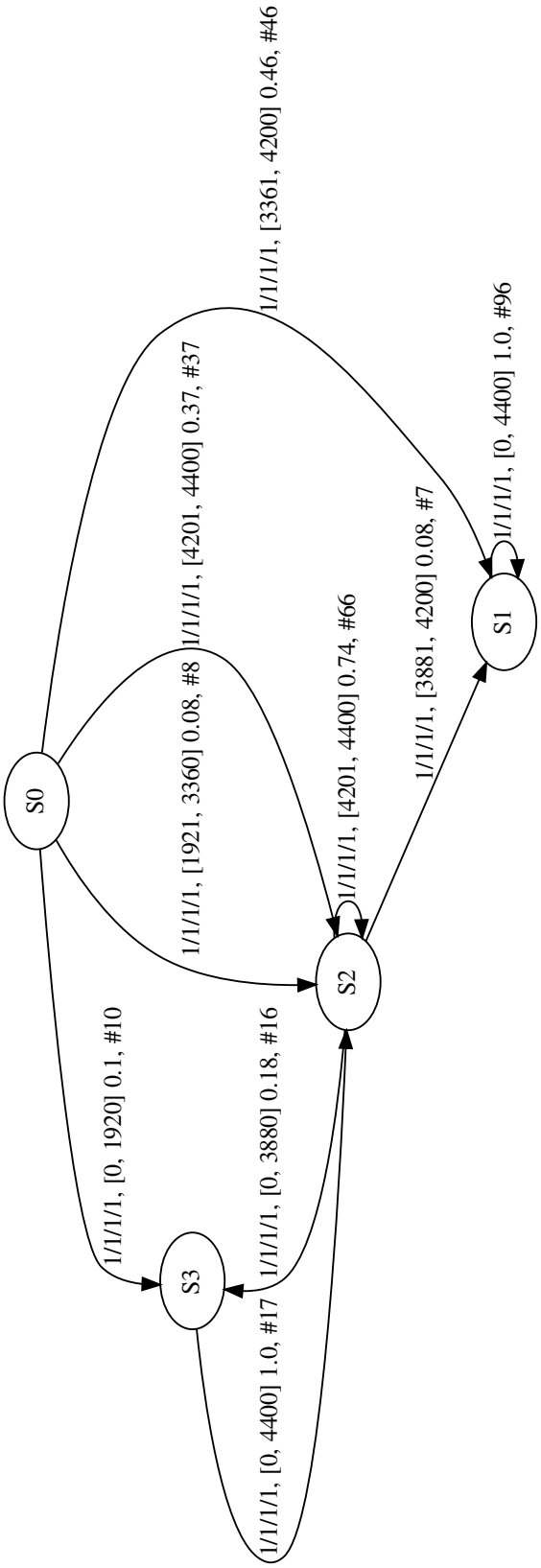

Figure 32. Trained model of subprocess 6.

March, 2020

BIBLIOGRAPHY

[1] N. Jazdi, "Cyber physical systems in the context of industry 4.0," in *2014 IEEE International Conference on Automation, Quality and Testing, Robotics*, IEEE, 2014.

[2] R. Baheti and H. Gill, "Cyber-physical systems," *The impact of control technology*, vol. 12, no. 1, pp. 161–166, 2011.

[3] S. McLaughlin, C. Konstantinou, X. Wang, L. Davi, A.-R. Sadeghi, M. Maniatakos, and R. Karri, "The cybersecurity landscape in industrial control systems," in *Proceedings of the IEEE*, vol. 104, IEEE, 2016.

[4] K. Stouffer, V. Pillitteri, S. Lightman, M. Abrams, and A. Hahn, "Guide to industrial control systems (ics) security," NIST, Tech. Rep., 2015. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-82r2.pdf.

[5] ENISA, *Critical infrastructures and services*. [Online]. Available: https://www.enisa.europa.eu/topics/critical-information-infrastructures-and-services/scada.

[6] C. Escudero, F. Sicard, and E. Zamai, "Process-aware model based idss for industrial control systems cybersecurity: Approaches, limits and further research," in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation*, IEEE, 2018.

[7] J. Giraldo, D. Urbina, A. Cardenas, J. Valente, M. Faisal, J. Ruths, N. O. Tippenhauer, H. Sandberg, and R. Candell, "A survey of physics-based attack detection in cyber-physical systems," *ACM Comput. Surv.*, vol. 51, no. 4, 76:1–76:36, 2018.

[8] D. Ding, Q.-L. Han, Y. Xiang, X. Ge, and X.-M. Zhang, "A survey on security control and attack detection for industrial cyber-physical systems," *Neurocomputing*, vol. 275, pp. 1674 – 1683, 2018.

[9] L. Zhang, Q. Wang, and B. Tian, "Security threats and measures for the cyber-physical systems," *The Journal of China Universities of Posts and Telecommunications*, vol. 20, pp. 25 –29, 2013.

[10] M. Caselli, E. Zambon, and F. Kargl, "Sequence-aware intrusion detection in industrial control systems," in *Proceedings of the 1st ACM Workshop on Cyber-Physical System Security*, ser. CPSS '15, Singapore, Republic of Singapore: ACM, 2015, pp. 13–24.

[11] R. Mitchell and I.-R. Chen, "A survey of intrusion detection techniques for cyber-physical systems," *ACM Computing Surveys*, vol. 46, 4 2014.

[12] T. Berg, B. Jonsson, M. Leucker, and M. Saksena, "Insights to angluin's learning," in *Proceedings of the International Workshop on Software Verification and Validation (SVV 2003)*, vol. 118, 2005, pp. 3 –18.

[13] A. Vodenčarević, H. K. Büning, O. Niggemann, and A. Maier, "Using behavior models for anomaly detection in hybrid systems," in *2011 XXIII International Symposium on Information, Communication and Automation Technologies*, 2011, pp. 1–8.

[14] T. Klerx, M. Anderka, H. K. Büning, and S. Priesterjahn, "Model-based anomaly detection for discrete event systems," in *2014 IEEE 26th International Conference on Tools with Artificial Intelligence*, 2014, pp. 665–672.

[15] O. Niggemann, B. Stein, A. Maier, A. Vodenčarević, and H. K. Büning, "Learning behavior models for hybrid timed systems," in *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, ser. AAAI'12, AAAI Press, 2012, pp. 1083–1090.

[16] A. Maier, O. Niggemann, M. Koester, and C. P. Gatica, "Automated generation of timing models in distributed production plants," in *2013 IEEE International Conference on Industrial Technology (ICIT)*, 2013, pp. 1086–1091.

[17] J. Goh, S. Adepu, M. Tan, and Z. S. Lee, "Anomaly detection in cyber physical systems using recurrent neural networks," in *2017 IEEE 18th International Symposium on High Assurance Systems Engineering (HASE)*, 2017, pp. 140–145.

[18] J. Inoue, Y. Yamagata, Y. Chen, C. M. Poskitt, and J. Sun, "Anomaly detection for a water treatment system using unsupervised machine learning," in *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, 2017, pp. 1058–1065.

[19] Q. Lin, S. Adepu, S. Verwer, and A. Mathur, "Tabor: A graphical model-based approach for anomaly detection in industrial control systems," in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, ser. ASIACCS '18, ACM, 2018, pp. 525–536.

[20] R. Medhat, S. Ramesh, B. Bonakdarpour, and S. Fischmeister, "A framework for mining hybrid automata from input/output traces," in *2015 International Conference on Embedded Software (EMSOFT)*, 2015, pp. 177–186.

[21] A. P. Mathur and N. O. Tippenhauer, "Swat: A water treatment testbed for research and training on ics security," in *2016 International Workshop on Cyber-physical Systems for Smart Water Networks (CySWater)*, 2016, pp. 31–36.

[22] S. Verwer, M. de Weerdt, and C. Witteveen, "A likelihood-ratio test for identifying probabilistic deterministic real-time automata from positive data," in *Grammatical Inference: Theoretical Results and Applications*, J. M. Sempere and P. García, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 203–216, ISBN: 978-3-642-15488-1.

[23] F. Aarts, H. Kuppens, J. Tretmans, F. Vaandrager, and S. Verwer, "Improving active mealy machine learning for protocol conformance testing," *Machine Learning*, vol. 96, no. 1, pp. 189–224, 2014.

[24] J. Goh, S. Adepu, K. N. Junejo, and A. Mathur, "A dataset to support research in the design of secure water treatment systems," in *Critical Information Infrastructures Security*, G. Havarneanu, R. Setola, H. Nassopoulos, and S. Wolthusen, Eds., Springer International Publishing, 2017.

[25] K. M. Aung, "Secure water treatment testbed (swat): An overview," Singapore University of Technology and Design, 2015.

[26] S. Adepu and A. Mathur, "An investigation into the response of a water treatment system to cyber attacks," in *2016 IEEE 17th International Symposium on High Assurance Systems Engineering (HASE)*, 2016, pp. 141–148.

[27] B. Steffen, F. Howar, and M. Merten, "Introduction to active automata learning from a practical perspective," in *Formal Methods for Eternal Networked Software Systems: 11th International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM 2011, Bertinoro, Italy, June 13-18, 2011. Advanced Lectures*, M. Bernardo and V. Issarny, Eds. Springer Berlin Heidelberg, 2011, pp. 256–296.

[28] S. Verwer, M. de Weerdt, and C. Witteveen, "Efficiently identifying deterministic real-time automata from labeled data," *Machine Learning*, vol. 86, no. 3, pp. 295–333, 2012.

[29] E. J. Keogh, S. Chu, D. M. Hart, and M. J. Pazzani, "An online algorithm for segmenting time series," in *Proceedings of the 2001 IEEE International Conference on Data Mining*, ser. ICDM '01, Washington, DC, USA: IEEE Computer Society, 2001, pp. 289–296, ISBN: 0-7695-1119-8.

[30] S. Verwer, *Rti+ tool*. [Online]. Available: http://www.cs.ru.nl/~sicco/software.htm.